Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

BIO-INSPIRED LEARNING AND HARDWARE ACCELERATION WITH EMERGING MEMORIES

by
Shruti R. Kulkarni

Machine Learning has permeated many aspects of engineering, ranging from the Internet of Things (IoT) applications to big data analytics. While computing resources available to implement these algorithms have become more powerful, both in terms of the complexity of problems that can be solved and the overall computing speed, the huge energy costs involved remains a significant challenge. The human brain, which has evolved over millions of years, is widely accepted as the most efficient control and cognitive processing platform. Neuro-biological studies have established that information processing in the human brain relies on impulse like signals emitted by neurons called action potentials. Motivated by these facts, the Spiking Neural Networks (SNNs), which are a bio-plausible version of neural networks have been proposed as an alternative computing paradigm where the timing of spikes generated by artificial neurons is central to its learning and inference capabilities. This dissertation demonstrates the computational power of the SNNs using conventional CMOS and emerging nanoscale hardware platforms.

The first half of this dissertation presents an SNN architecture which is trained using a supervised spike-based learning algorithm for the handwritten digit classification problem. This network achieves an accuracy of 98.17% on the MNIST test data-set, with about 4× fewer parameters compared to the state-of-the-art neural networks achieving over 99% accuracy. In addition, a scheme for parallelizing and speeding up the SNN simulation on a GPU platform is presented. The second half of this dissertation presents an optimal hardware design for accelerating SNN inference and training with SRAM (Static Random Access Memory) and nanoscale

non-volatile memory (NVM) crossbar arrays. Three prominent NVM devices are studied for realizing hardware accelerators for SNNs: Phase Change Memory (PCM), Spin Transfer Torque RAM (STT-RAM) and Resistive RAM (RRAM). The analysis shows that a spike-based inference engine with crossbar arrays of STT-RAM bit-cells is 2× and 5× more efficient compared to PCM and RRAM memories, respectively. Furthermore, the STT-RAM design has nearly 6× higher throughput per unit Watt per unit area than that of an equivalent SRAM-based (Static Random Access Memory) design. A hardware accelerator with on-chip learning on an STT-RAM memory array is also designed, requiring 16 bits of floating-point synaptic weight precision to reach the baseline SNN algorithmic performance on the MNIST dataset. The complete design with STT-RAM crossbar array achieves nearly 20× higher throughput per unit Watt per unit mm^2 than an equivalent design with SRAM memory.

In summary, this work demonstrates the potential of spike-based neuromorphic computing algorithms and its efficient realization in hardware based on conventional CMOS as well as emerging technologies. The schemes presented here can be further extended to design spike-based systems that can be ubiquitously deployed for energy and memory constrained edge computing applications.

BIO-INSPIRED LEARNING AND HARDWARE ACCELERATION WITH EMERGING MEMORIES

by
Shruti R. Kulkarni

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology – Newark
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Helen and John C. Hartmann Department of Electrical and Computer Engineering

December 2019

APPROVAL PAGE

BIO-INSPIRED LEARNING AND HARDWARE ACCELERATION WITH EMERGING MEMORIES

Shruti R. Kulkarni

Dr. Bipin Rajendran, Dissertation Advisor — Date
Associate Professor, Department of Electrical and Computer Engineering, New Jersey Institute of Technology

---

Dr. Ali N. Akansu, Committee Member — Date
Professor, Department of Electrical and Computer Engineering, New Jersey Institute of Technology

---

Dr. Durgamadhab Misra, Committee Member — Date
Professor, Department of Electrical and Computer Engineering, New Jersey Institute of Technology

---

Dr. Horacio G. Rotstein, Committee Member — Date
Professor, Math Bio & Computational Neuroscience, Biological Sciences, New Jersey Institute of Technology

---

Dr. Osvaldo Simeone, Committee Member — Date
Professor, Information Engineering, King's College, London

BIOGRAPHICAL SKETCH

Author: Shruti R. Kulkarni

Degree: Doctor of Philosophy

Date: December 2019

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, USA, 2019
- Master of Technology in Electrical Engineering,
  Indian Institute of Technology Bombay, Mumbai, India, 2012
- Bachelor of Engineering in Electronics and Communication Engineering,
  RNS Institute of Technology, Bengaluru, India, 2010

Major: Electrical Engineering

Presentations and Publications:

S. R. Kulkarni, B. Rajendran, "Spiking Neural Networks for Handwritten Digit Recognition – Supervised Learning and Network Optimization," *Neural Networks*, vol. 103, pp. 118-127, July 2018.

S. R. Nandakumar, S. R. Kulkarni, A. V. Babu, B. Rajendran, "Building Brain-Inspired Computing Systems: Examining the Role of Nanoscale Devices," *IEEE Nanotechnology Magazine*, vol. 12, no. 3, pp. 19-35, Sept 2018.

S. R. Kulkarni, D. V. Kadetotad, S. Yin, J-s. Seo, B. Rajendran, "Neuromorphic Hardware Accelerator for SNN Inference based on STT-RAM Crossbar Arrays," accepted at *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genova, Nov 2019.

S. R. Kulkarni, D. V. Kadetotad, J. Seo, B. Rajendran, "Well-Posed Verilog-A Compact Model for Phase Change Memory," *Proceedings of International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pp. 369-373, Austin, TX, Sept 2018.

S. R. Kulkarni, A. V. Babu, B. Rajendran, "Acceleration of Convolutional Networks Using Nanoscale Memristive Devices," *Proceedings of 19th International Conference on Engineering Applications of Neural Networks (EANN)*, pp. 240–251, Springer, Cham, Bristol, U.K., Sept 2018.

S. R. Kulkarni, J. M. Alexiades, B. Rajendran, "Live Demonstration: Image Classification Using Bio-inspired Spiking Neural Networks," *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-1, Florence, May 2018.

S. R. Kulkarni, J. M. Alexiades, B. Rajendran, "Learning and Real-Time Classification of Hand-written Digits with Spiking Neural Networks," *Proceedings of 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 128-131, Batumi, Georgia, Dec 2017.

S. R. Kulkarni, A. V. Babu, B. Rajendran, "Spiking Neural networks – Algorithms, Hardware Implementations and Applications," *Proceedings of 60th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Invited paper, pp. 426-431, Boston, MA, Aug 2017.

S. R. Kulkarni, B. Rajendran, "Scalable CMOS Digital Architecture for Spike Based Supervised Learning", *Proceedings of 16th International conference on Engineering Applications of Neural Networks (EANN)*, CCIS 517, pp. 149 - 158, Island of Rhodes, Greece, Sept 2015.

C. Shetty, S. Nitchith, R. Rawat, S.R. Nandakumar, P. Shah, S. Kulkarni, B. Rajendran. "Live demonstration: Spiking Neural Circuit Based Navigation Inspired by C. Elegans Thermotaxis," *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1905-1905, Lisbon, Portugal, May 2015.

Dedicated to my parents who always motivate me to explore the unknown.

ACKNOWLEDGMENT

I take this opportunity to thank all the people without whose support I would not have reached so far in my research work. First of all, I thank my advisor Dr. Bipin Rajendran for providing me with the opportunity to work in the exciting area of neuromorphic engineering. It is because of his invaluable guidance and discussions that I have been able to push myself to reach the end of my Ph.D. research.

I also to thank Dr. Ali N. Akansu, Dr. Osvaldo Simeone, Dr. Horacio G. Rotstein, and Dr. Durgamadhab Misra for having kindly agreed to be part of my dissertation committee and providing me with their invaluable feedback that has improved this dissertation.

I also acknowledge the funding agency, Semiconductor Research Corporation (SRC) for having funded my research work on the non-von Neumann hardware design with Non-Volatile Memory technologies. I also gratefully acknowledge our collaborators on this project from the Arizona State University, Dr. Jae-sun Seo, Shihui Yin and Deepak V. Kadetotad. I also acknowledge the Teaching Assistantship funding from the department of Electrical and Computer Engineering (ECE) during my first two years at NJIT. The first one and a half years of my Ph.D. research were completed at the Indian Institute of Technology Bombay, where I was supported by the Teaching Assistantship from the department of Electrical Engineering and Intel fellowship. I also take this opportunity to thank all the faculty members from IIT Bombay who helped in both technical and non-technical issues I faced in the initial year of my research programme and in developing my scientific temperament. Most importantly, I would also like to mention all my colleagues and friends at IIT Bombay who have also played a significant role in my research journey.

Last, but not the least, I also take this opportunity to thank my family, friends and colleagues who have been constantly supporting me throughout my

journey of Ph.D. I also gratefully acknowledge the valuable discussions with my colleagues at the Intelligent Computing Lab, Anakha V. Babu, Alireza Bagheri, Bleema Rosenfeld, Nandakumar S. R. and Vinay Joshi. I also thank the interns and undergraduate students who worked with me at our lab during the summers, and have also contributed to some of the work in this dissertation. The staff and faculty from the ECE department, NJIT have also been very helpful and supportive throughout my Ph.D. program. I also acknowledge the dissertation document review feedback from Ms. C. Gonzalez and Dr. G. Ziavras from the office of Graduate Studies, NJIT and also in helping me out with the degree completion paperwork.

In short, my Ph.D. dissertation would not have been successfully completed had it not been for the direct or indirect contributions of the faculty, staff, colleagues, friends, and family.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation and Overview

Cognitive processing capabilities of the human brain are far superior than any supercomputer till date. Apart from their processing ability, they are extremely efficient in the number of operations being performed, with some neuro-biological studies showing that the brain is able to carry out close to $100,000$ billion FLOPS per unit of Watt, while today's latest supercomputer can achieve only around 10 billion FLOPS per unit Watt [1, 2]. The superior computational efficiency of biological systems has inspired the quest to reverse engineer the brain in order to develop intelligent computing platforms that can learn to execute a wide variety of data analytics and inference tasks [3]. However, in spite of several decades of research, the principles of information processing in the brain is not yet fully understood, and efforts to mimic its power efficiency and fault-tolerance in computing systems remain unfulfilled. The ongoing quest towards understanding the functioning of the brain has propelled research towards reverse engineering the brain for various neurological studies, through computational neuroscience techniques [4]. Additionally, the field of artificial intelligence has also taken inspiration from the neural architecture of the brain, with the result of Deep Learning framework emerging as the state-of-the-art for various cognitive and data processing tasks. In particular, inspired by the Nobel prize winning work of Hubel and Weisel on elucidating the mechanisms of information representation in the visual cortex [5], multi-layer convolutional neural networks have shown impressive performance for a wide variety of applications such as image recognition, natural language processing, speech recognition and video analytics [6–15].

The field of neuromorphic engineering tries to mimic the key computational aspects of biological computations in silicon circuits. Earliest works by Carver Mead at Caltech describe sub-threshold VLSI circuit designs mimicking the neuron's membrane potential dynamics [16]. During the last decade, there have been a number of demonstrations of sensory data acquisition systems inspired by the cochlea, retina, etc. [17–20]. To make neural networks more biologically plausible, a new generation of learning models have emerged called the spiking neural networks (SNNs). These models mimic event based data processing and communication aspects of biological neurons. Neuro-biological studies have also shown that the rate of spikes in the brain is very small, 0.1 to 300 Hz [21]. This low operational firing rate has been postulated to be one of the reasons for the brain's high energy efficiency.

Inspired by this brain-inspired asynchronous computing paradigm, several neuromorphic hardware designs have been demonstrated with significant computational efficiency improvements [22–25]. These neuromorphic SNN accelerator architectures closely integrate the memory and logic units in order to minimize the cost of data transfers associated with conventional von Neumann architectures. However, these digital hardware platforms make use of on-chip SRAM to store the network parameters which has limited density. For running larger neural network models, these platforms have to make use of larger density DRAMs (which are primarily off-chip). Additionally, DRAMs are slower in access and consume more power as they need to be refreshed periodically. For building efficient and compact hardware accelerators, there is a need for memory technologies with smaller form factor and faster accesses.

Research over the last decade has led to the development of several nanoscale non-volatile memory devices such as phase change memory (PCM), spin-transfer torque RAM (STT-RAM), and resistive RAMs (RRAM), ideally suited for in-memory or near-memory computing with a crossbar based architecture. The conductance

states of these devices could be used to represent synaptic weights of large neural networks. Design studies suggest that memristive neural network accelerators with analog memory storage can potentially achieve significantly higher throughput compared to GPUs, provided several challenges typically associated with nanoscale devices can be addressed [26–33]. In particular, the crucial challenges include the variability and stochasticity associated with the device conductance, and the additive noise from the peripheral circuitry such as Analog to Digital converters (ADCs) and Digital to Analog Converters (DACs) [26, 34–36].

In this dissertation, we start by discussing the principles to encode and train a system using just binary spikes, akin to the action potentials seen in the brain. We explore two of the existing spike based supervised learning algorithms - Remote Supervised Method (ReSuMe) and the Normalized Approximate Descent (NormAD), in terms of optimizing them for efficient hardware realizations. As the NormAD algorithm is shown be be faster at its convergence, we apply this rule to train an SNN for solving the problem of handwritten digit classification [37]. We also show that this spike based algorithm performs similar to an equivalent network with non-spiking neurons (ANNs) in terms of its classification accuracy. We also discuss the various insights from our network training and optimization studies for an eventual efficient implementation. This part of the research was carried out by developing a CUDA (Compute Unified Device Architecture) simulator to be run on a GP-GPU (General Purpose Graphical Processing Unit) card.

Building better hardware for realizing SNNs has also been emphasized by the research community, as is demonstrated by the chips of IBM, Intel, etc. [22, 23, 25]. In this direction, we present the basic units of designing an ASIC (Application Specific Integrated Circuit) for hardware acceleration of SNNs, which forms the second part of this dissertation. The key studies and contributions of this work are as following:

1. Network optimization in terms of spike based temporal features for a supervised learning algorithm, and their comparison with existing class of networks.
2. Architectural framework using existing and emerging memory devices for realizing spike based learning on hardware.
3. Development of mathematically well-posed compact models for non volatile memory (NVM) devices with reliability features.
4. Hardware design and throughput comparison of STT-RAM based accelerator for SNNs.

1.2 Organization of the Dissertation

The research work described in this dissertation is comprised of two parts. The first part focuses on SNN algorithms and second part on the hardware design.

Chapter 2 presents a background on bio-inspired computing and the learning mechanism observed in neuro-biological studies. We then present the SNN developed for solving the handwritten digit classification problem in Chapter 3. We also present CUDA based software acceleration of this SNN on a GP-GPU.

Chapter 4 presents the compact models for the NVM devices that we built and use in our further architecture development and analysis. We also present the process of introducing conductance variabilities and faults in the model simulation. We discuss the design of three models: Phase Change Memory (PCM), Spin Transfer Torque RAM (STT-RAM), and Resistive RAM (RRAM) devices.

In Chapter 5, we present the non-von Neumann architecture design for accelerating neural networks, specifically the SNNs. Here, we discuss a CMOS (Complementary Metal-Oxide Semiconductor)-based digital design for an SNN along with the spike based Remote Supervised Method (ReSuMe) training rule [38]. We then present the prospects of using an NVM crossbar array with near-memory compute approach for accelerating SNNs. We explore the analog (multiple levels in a single device) and digital (two levels within a device) storage arrays with NVM devices. For the digital NVM array, we compare the design realization with PCM,

RRAM, and STT-RAM memory arrays. Finally, we present the STT-RAM based design for learning and inference, and also compare its inference performance with an equivalent SRAM based design. Chapter 6 summarizes and concludes the dissertation. We also present a discussion on the future trends in this area of research and the challenges that need to be addressed.

Appendix A gives the details of the ReSuMe and NormAD algorithm parameters and the simulation procedures. Appendix B lists the parameters used in developing the compact models for the three NVM devices (PCM, STT-RAM, and RRAM) discussed in this dissertation in Chapter 4.

CHAPTER 2

BIO-INSPIRED COMPUTING AND HARDWARE DESIGN

Nature has always been a inspiration for various fields of engineering, ranging from mechanical to computing [39]. Studies on the human visual cortex system have inspired the development of deep convolution neural networks (CNNs) [5, 6]. The ground breaking work by Yann LeCun on training CNNs [6], followed by the ILSVRC-2012 challenge winning CNN by Alex Krizhevksy [40] has led to the development of many improved network architectures and training techniques to develop deep networks with super-human level accuracy [7, 9–13, 41].

Parallel to artificial neural networks, the third generation of neural networks called Spiking Neural Networks (SNNs) have also been emerging as a potential computing framework to solve machine learning problems. Similar to the time-based information encoding using action potential transmitted by nerve cells, SNNs operate by making use of precise timings of all-or-none spikes transmitted between neurons [42, 43]. These networks, which are capable of employing the temporal dimension through memory based neuronal dynamics and synaptic delays, have been successfully shown to emulate the different spike-firing dynamics in the brain [44].

Throughout this dissertation, we use the term artificial neural networks (ANNs) to denote the non-spiking networks. The background details of these two different types of neural networks will be discussed in the following sections.

2.1 Artificial Neural Networks

The earliest work done towards devising neuron models for computing, abstractly based on the functioning of the biological neural network was the McCulloch-Pitts model of the neuron (also referred to as the first generation of neural networks) [45]. This model used a threshold gate (also called a perceptron) as a neuron, wherein

the neuron had a high output if the weighted input sum exceeded the threshold. This model was able to capture the all-or-none firing dynamics of the neuron based on simple thresholding. With just digital inputs and binary outputs, they could be inter-connected to realize any Boolean function. However, these models were limited in their functionality as their activation function was non-differentiable and hence, could not be trained autonomously. Recently, there has been an emergence of such models of neurons as a potential low memory and compute foot-print networks [46], through the use of straight-through estimator function [47]. More details on their usage will be discussed in the following section and upcoming chapters.

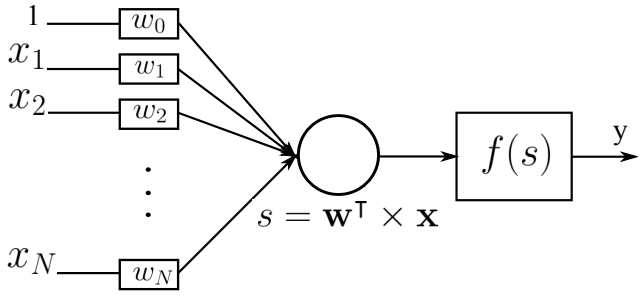

Figure 2.1 Model of a neuron used in second generation ANNs. The neuron shown above computes the weighted sum of all its inputs in s, which is then applied to an activation function f to generate the final output. The neuron additionally has a bias line (of weight w_0) for which the input is kept at unity.

The second generation of neuron models and architectures supported continuous valued inputs and outputs, and hence can act as universal function approximators. The basic scheme of these neurons is as shown in Figure 2.1. A neuron consists of N inputs, which are individually weighted and summed up; the result is then passed to an activation function to generate the output. Activation functions $f()$, such as sigmoid, tanh, softmax, ReLU (Rectified Linear Units), etc. have been used in different implementations of the ANNs. The continuous and hence, differentiable nature of the second generation of neural networks also led to the development of various optimization techniques to get the desired outputs, through the application of chain rule of derivatives, what is commonly called the back-propagation rule.

These models of neurons have been widely used in various ANN architectures, ranging from simple fully connected, convolution, and recurrent neural networks. Amongst these, convolutional neural networks have established themselves as the state-of-the-art in image classification, and object detection problems [41,48]. One of the earliest CNNs, LeNet-5, which is a six layer deep network, with five convolution layers, showed an error of 0.95% on the MNIST (Modified National Standards and Technology database) test data-set (with $60,000$ training images and $10,000$ test images of handwritten digits) [6, 49]. This architecture consists of two sets of alternating layers of convolution and sub-sampling that extract the features and perform an averaging to achieve translation invariance of the input. The last layer which is the fully-connected layer is the classifier layer and predicts the label of the input. This architecture has been the baseline for many of the current deep CNNs which have improved over time with the incorporation of techniques such as max-pooling instead of sub-sampling, dropout of connections between layers, etc. The deep CNNs have also been successfully employed in various applications apart from image, such as, Alpha Go game by Google's Deep Mind [50], drug discovery by AtomNet [51], speech recognition [10, 14], medical image analysis [52], etc.

2.2 Bio-inspired Computing

The neurons in the ANN discussed above, implement memoryless non-linear transformation of the input signals to create real-valued output signals. This is vastly different from the behavior of neurons in the brain, which encode information in the time of issue of binary signals called action potentials (or spikes) based on the time of arrival of incoming spike signals from upstream nodes in the network. SNNs have been demonstrated to have a higher computational power owing to the time information embedded in the spike signals [42,53–55]. The spikes transmitted by these neurons are fixed amplitude signals occurring over a period of several time-steps. SNNs have also

shown impressive performance in various applications such as automatic navigation control [56–58], neuroprosthetic controllers [59], pattern recognition [60–62], etc. The development of SNNs seems highly promising in terms of the energy efficiency. As mentioned in the previous chapter, the brain is able to perform nearly $10,000\times$ more operations per unit Watt than the fastest super-computer available today [1].

The basic computing units of the SNNs are the spiking neurons. There are various types of spiking neuron model developed by various groups, where the detailed description of the neuronal dynamics is based on the intricate interaction of ion-channels on the cell membrane. In the following subsections we present the basics of the operations of biological neurons and the various levels of abstractions in mathematical models of these neurons and synapses.

2.2.1 Spiking Neuron Models

A biological neuron consists of the cell body (the nerve cell), synapses that act as links between two neurons, axons which transmit the signal out of the neuron and dendrites which are the inputs of the neurons [63]. At the cellular level, the dynamics of different ion-specific channels on the neuron's membrane causes them to open or close, leading to inflow or outflow of certain ions. At rest, the concentration of these ions in the intra-cellular and extra-cellular fluid maintains an equilibrium across the neuron's membrane, with the potential referred to as the Nernst potential [4]. When the neuron receives incoming current from its synapse the ion channels of Sodium (Na^+) open and an influx of Na^+ ions takes place, which can potentially set up a positive feedback and rapidly raise the membrane potential value (depolarization). This creates an action potential across the membrane. This process is followed by closing of Na^+ ion-channels and subsequent opening of K^+ ion-channels causing the potential to lower and become more negative (or hyper-polarized) for a short period of time. The transient dynamics of these ion-channels also imposes a certain refractory

period on the membrane potential, wherein it remains at its resting value (typically around $-70\,\text{mV}$) for a certain period of time.

The above mentioned membrane potential dynamics is typically abstracted as an integrated value of the incoming synaptic current which is reset to its resting potential when this value exceeds a threshold, with an action potential being issued thereafter. The transient rise and fall of the membrane are modeled by having a capacitance and a resistance along with a leak path in the neuron membrane. There are several mathematical models of biological neurons capturing various levels of details of their electrical dynamics.

The earliest detailed model capturing the dynamics of different ion channels on the neuron membrane was proposed by Hodgkin and Huxley in 1952 [64]. The basic equation describing the membrane potential V dynamics is,

$$I = C_M \frac{dV}{dt} + I_i \tag{2.1}$$

where I is the net membrane current, C_M is the membrane's capacitance, and I_i is the ionic current density, which is a cumulative of all the ions flowing in and out of the membrane (Na^+, K^+, Cl^-, etc.). The generic form of the ionic current for each of these ions i is given as,

$$I_i = g_i(V - V_i) \tag{2.2}$$

Here, V_i is the displacement of the respective ionic Equilibrium potentials (E_{Na}, E_K and the leak potential E_l [43]) from the membrane's resting value E_r. The above mathematical descriptions of spiking neuron were based on the observations from various voltage clamp experiments for measuring the surface currents from a giant nerve fiber. It was also observed that the ionic conductances g_i, are also functions of the membrane potential and continuously keep evolving over a period of time. The

Potassium (K^+) conductance g_K can be described as,

$$g_K = \bar{g}_K n^4, \tag{2.3}$$

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n, \tag{2.4}$$

Similarly, the sodium channel conductance are given by,

$$g_{Na} = m^3 h \bar{g}_{Na}, \tag{2.5}$$

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m, \tag{2.6}$$

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h \tag{2.7}$$

Here the respective αs and βs are time-dependent rate constants. n, m and h are dimensionless state variables (denoted as x), where the first term $(1-x)$ represents the proportion of activating ions on the outside of the membrane and x represents the proportion inside the membrane [64]. The increase and decrease of each of these state variables x recreates the membrane potential dynamics of issuing an action potential (or spike) and also models the refractory period. Thus, the Hodgkin-Huxley's neuron model captures a detailed description of the neuronal dynamics from the different intra- and extra-cellular ions.

While the above detailed Hodgkin-Huxley (HH) model uses four-dimensional equation, there are several models which reduce the HH model to two-dimensional form primarily to reduce the computational complexity while still being able to capture the different spiking dynamics [43, 65]. This becomes important when simulating large networks of spiking neurons. The reduction from the 4D HH model is achieved by performing the phase plane analysis, which involves a projection of the variables (m, n, and h) onto a 2D space [43]. Models such as the Morris-Lecar, FitzHugh-Nagumo, Izhikevich, etc. describe the neuron using just two state variables, the membrane potential v and a recovery variable u [66–68].

The 2D spiking neuron model proposed by Izhikevich can be tuned to recreate the observed dynamics of various cortical neurons [68]. The basic differential equations of this model are:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.8}$$

$$u' = a(bv - u) \tag{2.9}$$

Here v represents the membrane potential and u the membrane recovery variable. If $v \geq 30\,\text{mV}$, then $v \leftarrow c$ and $u \leftarrow u + d$ which models the resetting of the membrane potential and the recovery variable after spike is issued. The parameters a, b, and c control the time-scale and sensitivity of the variables u and v.

Another 2D model of spiking neuron developed from the electrophysiological studies of a nerve fibre is the adaptive exponential integrate-and-fire (aEIF) model [69]. This model incorporates an exponential adaptation for the membrane potential to allow for a smooth spike initiation within the neuron. It consists of a membrane potential variable V and an adaptation variable w, and the dynamics are given as:

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right) - w + I \tag{2.10}$$

$$\tau_w\frac{dw}{dt} = a(V - E_L) - w \tag{2.11}$$

Here I is the input current, C the membrane capacitance, g_L the leak conductance, E_L the leak reversal potential, V_T the threshold, Δ_T the slope factor determining the sharpness of the spike, a the adaptation coupling parameter and δ_w is the adaptation time constant. The membrane potential is artificially reset to its reset value V_r when the potential V exceeds the threshold. The adaptation variable w is also changed by a finite amount b when a spike is issued, i.e., $w \leftarrow w+b$. Similar to the Izhikevich model, the aEIF model also captures the firing dynamics observed in the different types of cortical neurons. However, the difference from the Izhikevich model is the inclusion of exponential adaptation, which makes the rise of membrane potential rapid during

a spike initiation (which is also observed in cortical neurons). Thus, the aEIF spiking neuron model captures the behavior of a biological neuron more closely while still being relatively simple in terms of computation requirements than the HH model.

The simplest and a highly abstracted model of a spiking neuron described by just a one dimensional equation is the leaky-integrate-and-fire (LIF) model [70]. This model captures the neuron dynamics by representing the membrane potential as a voltage across a capacitor which is charged by incoming input currents, connected in parallel with a leaky conductance path. A generalization of this LIF model is the Spike Response Model (SRM), where all the model parameters are time-dependent [71]. SRM is a 1D model of a spiking neuron, where the dynamics of the membrane potential u is a summation of different kernels, η, ϵ and κ, each capturing the neuron's response to its own issued spike, incoming spike and incoming current, respectively. The SRM model is given by the following equation [43],

$$u_i(t) = \eta(t - \hat{t}_i) + \sum_j w_{i,j} \sum_f \epsilon_{ij}(t - \hat{t}_i, t - \hat{t}_j^{(f)}) + \int_0^\infty \kappa(t - \hat{t}_i, s).I^{ext}(t - s).ds \quad (2.12)$$

In the above equation it is assumed that the neuron under consideration i, has spiked at time $\hat{t}_i$. Here, $w_{i,j}$ represents the synaptic weight between pre-synaptic neuron j and post-synaptic neuron i. The neuron's threshold also varies with time which is captured by the function $\eta()$. This model is also able to re-create the behaviors of different types of neurons.

While the above described models are suitable for experiments trying to mimic the cortical behavior, for most practical machine learning applications, the simplest LIF model suffices while being used in Spiking Neural Networks. LIF model is a special simplified case of the SRM model. We now present the LIF model used in the experiments discussed in this dissertation. The membrane potential $V(t)$ evolves according to the following differential equation,

$$C\frac{dV(t)}{dt} = -g_L(V(t) - E_L) + I_{syn}(t) \quad (2.13)$$

The membrane potential is artificially reset to its resting value E_L when $V(t) \geq V_T$, and a spike is issued to the downstream synapse. The incoming spikes (represented

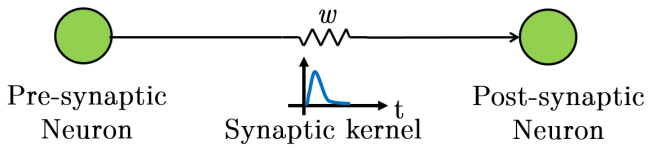

Figure 2.2 A simple case of two neurons connected by a synapse of strength 'w'. The synapse transforms the incoming spike into an equivalent current, through a kernel, which is presented as input to the post-synaptic neuron.

by a δ function) on a synapse, generate the post-synaptic current ($I_{syn}(t)$) as,

$$c(t) = \sum_i \delta(t - t^i) * \left(e^{-(t-t^d)/\tau_1} - e^{-(t-t^d)/\tau_2} \right) \tag{2.14}$$

$$I_{syn}(t) = w \times c(t) \tag{2.15}$$

where t^i are the time instants at which a spikes were issued. t^d represent the synaptic transmission delay for the spikes. Here, the term $c(t)$ is the synaptic kernel, which performs filtering on the incoming spikes. Figure 2.2 shows a simple network of two spiking neurons connected by a synapse with a double decaying exponential kernel. The evolution of the membrane potential with every incoming spike from pre-synaptic neurons can be seen Figure 2.3.

Spikes from the pre-synaptic neuron are transformed into post-synaptic current by the synaptic kernel as in Equation 2.15. Figure 2.3 show the evolution of post-synaptic current and the membrane potential of the post-synaptic neuron.

There is a non-linear dependence between the time of issue of spikes of a neuron and the incoming spike times, due to the weighted summation, integration and reset. Figure 2.4 illustrates a particular non-linearity associated with the LIF neuron which is excited by a DC current lasting 100 ms – the output spike frequency in that interval is a non-linear function of the excitation current. Typically, for the spiking neurons on the input layer, the sensory signals can be directly applied as input currents.

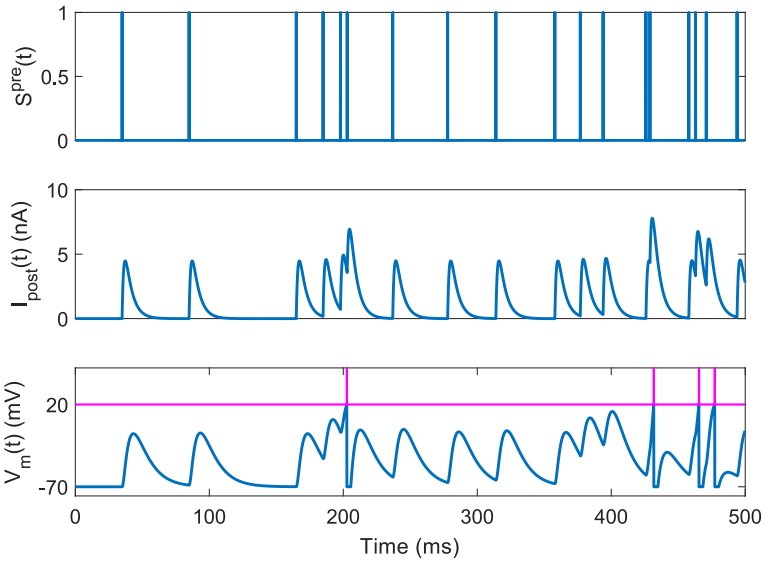

Figure 2.3 Spikes input to a neuron (top). Evolution of the synaptic current (middle) and membrane potential (bottom). The membrane potential at point where the threshold is crossed is artificially set to a higher value (40 mV), for the sake of clarity.

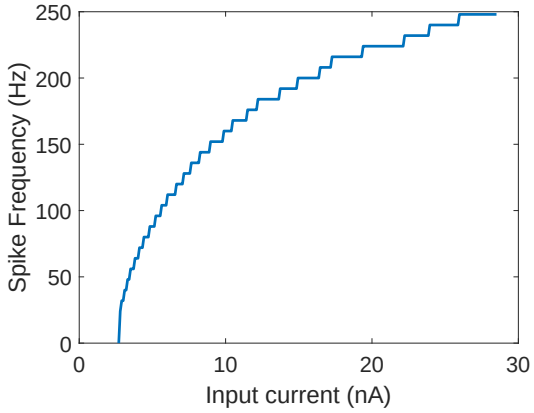

Figure 2.4 Spike frequency of an LIF neuron excited by a constant input current exhibits a strong non-linear dependence.

While we use the above described LIF spiking neuron model to carry out the spike based algorithmic studies, for realizing SNNs on hardware, there are several works that further simplify the neuron model by eliminating its leak term, making it an integrate and fire model (IF) [46, 62]. In the later part of this dissertation, where we discuss the hardware design for accelerating SNNs, we make use of a similar IF model called Binary Activation (BA) neuron [46]. This model is inspired by the

Perceptron model of an artificial neuron which was proposed by Frank Rosenblatt in 1958 [72], with the neuron's output being a '1' or a '0' depending on the value of the accumulated weighted inputs. This model does not use a leak term and is suitable for problems that do not require precisely timed spikes to be generated by the neuron. The activation function of a neuron j at timestep t is given by:

$$a_j(t) = y_b \left(\sum_{i=1}^{N} a_i(t) w_{i,j} + b_j \right), \tag{2.16}$$

where y_b is the threshold function, with $y_b(x) = 1$ only if $x > \theta$ [46]. Here, θ is the membrane potential threshold. The term within the brackets represents the spiking neuron's membrane potential $v(t)$. The membrane potential is reset every time-step and a new potential value is evaluated. The neuron j receives incoming spikes from each of the pre-synaptic neurons i, with synaptic weights $w_{i,j}$ between pre-synaptic neuron i and post-synaptic neuron j. Each neuron j also has a bias b_j.

An extension of this neuron for time varying inputs is also presented in [46]. In this model (also referred to as continuous integration model), the membrane potential is an integrated value over several time-steps. On exceeding the threshold, the membrane potential is reduced by an amount equal to θ and a spike $a(t)$ is issued. The model equations are as given below [46]:

$$v_j(t) = \sum_{i}^{m} (a_i(t) w_{i,j}) + b_j^k + v_j(t-1), \tag{2.17}$$

$$a_j(t) = y_b(v_j(t^-)), \tag{2.18}$$

$$v_j(t) = v_j(t^-) - \theta . a_j(t). \tag{2.19}$$

It can be seen from the above expressions that the Binary Activation neuron's output $a^k(t)$ is discontinuous in nature. For using this model in SNNs that can be trained using gradient descent for a particular task, the membrane potential $v^k(t)$, is linearized near the region of the threshold by making use of a straight-through

gradient estimator [47], given as,

$$g_j(v_j(t)) = \begin{cases} (1/2\theta), (\theta - 1) \leq v_j(n) \leq (\theta + 1) \\ 0, \ otherwise \end{cases} \tag{2.20}$$

Further details on the use of this neuron model and the Binary Activation SNN (BASNN) while designing neurosynaptic core will be discussed in Chapter 5.

Having seen the basics of the spiking neuron and inter-neuron communication, we now discuss the learning rules for SNNs using the LIF neuron model and exponentially decaying synaptic kernels. The learning rules discussed in this chapter are for training the spiking neuron to create spikes at the desired times. The learning rule for BASNN will be discussed in detail in Chapter 5.

2.2.2 Spike-based Learning Rules from Biology

Most of the learning rules that have been postulated from biology are based on the firing activities of the neurons and the precise times of these neurons. One of the most prominent learning rule from neuro-biological studies was proposed by Donald O. Hebb, who postulated that the strength of the synaptic connection between two neurons is dependent on their activities or spiking rates (ν_i) [73]. The Hebbian law states that, "Neurons that fire together wire together". Mathematically, the weight change for a synapse between a pre-synaptic neuron i and post-synaptic neuron j is given as,

$$\Delta w \propto \nu_i \nu_j \tag{2.21}$$

However, the drawback to this rule was that there was no mechanism to bound the weights. Later, a modification to this rule was proposed, called Spike Timing Dependent Plasticity (STDP) [74]. As per this rule, the weights are updated according to the precise timings of the pre- (t_{pre}) and post- (t_{post}) synaptic neurons, in a learning

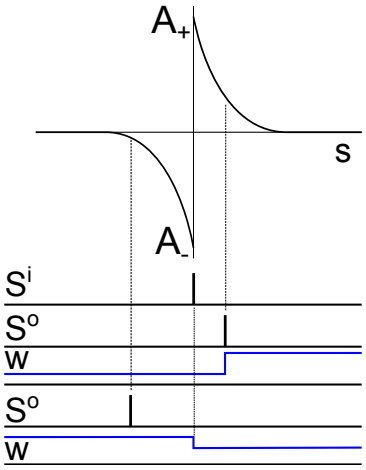

Figure 2.5 Weight modification in Spike Timing Dependent plasticity (STDP).

window as shown in Figure 2.5. Mathematically, the learning window is modeled as [75],

$$W(s)^{STDP} = \begin{cases} +A_+ \exp\left(\frac{-s}{\tau_+}\right) & \text{if } s \geq 0 \\ -A_- \exp\left(\frac{s}{\tau_-}\right) & \text{if } s < 0 \end{cases} \tag{2.22}$$

where $s = t_{post} - t_{pre}$. The STDP rule has been applied in SNNs solving the problem of handwritten digit classification and also as a pre-processing step in deep SNNs [61, 76–82].

2.2.3 Supervised Learning in SNNs

Though the above mentioned rules capture the key aspects of synaptic strength modification based on spike timings, but they are not suitable for tasks such as classification or recognition. The goal of supervised learning problem in SNNs is to be able to transform a set of incoming spike trains to a desired set of spike train at the output of a spiking neuron (Figure 2.6).

One of the earliest supervised learning rule in SNNs is the Remote Supervised Method (ReSuMe) [38], that uses the local STDP rule for weight update [74]. Other spike based learning algorithms that have been proposed include the SpikeProp

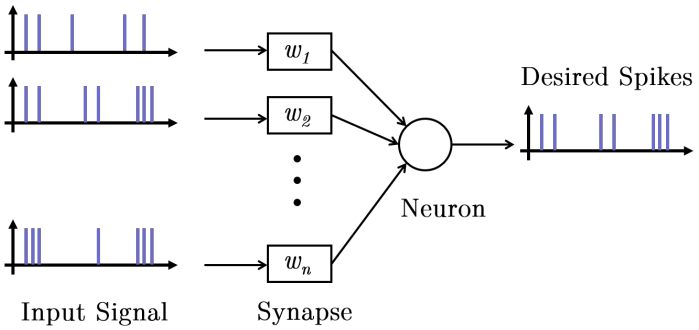

Figure 2.6 Goal of supervised learning is to determine the set of weights **w** that transform the incoming spike trains into the desired spike train shown.
Source: [37].

algorithm (restricted to single spike learning) [83], SPAN and PSD (which applied the learning rule to the analog version of the spike trains) [84, 85]. A variant of ReSuMe algorithm, called the Delay Learning (DL)-ReSuMe, in addition to the synaptic weights, made use of the transmission delays of synapses interconnecting the neurons as parameters to train the network [86]. This algorithm has been shown to be superior in terms of accuracy and speed of convergence compared to the basic ReSuMe algorithm. The accurate synaptic efficiency adjustment method is another spike-error triggered supervised learning rule based on STDP, which optimizes a cost function defined in terms of membrane potential differences [87]. This method has been used to demonstrate excellent performance in several UCI datasets with few training parameters. The Synaptic Kernel Inverse Method (SKIM) [88], evaluates the weights analytically rather than learning them iteratively and has been applied to the problem of speech based digit recognition in a small network with 50 neurons. Based on the SKIM method, the convex optimized synaptic efficiencies (CONE) algorithm was developed [89] and was used for the problem of gait detection. The generalization capability of this algorithm and the noise tolerance of a variation of the algorithm called CONE-R has also been demonstrated. There are other recent algorithms that train the network with spike based rules, but do not consider the temporal dimension while training [46, 90].

Besides the above-mentioned approaches for designing learning algorithms for SNNs that operate directly in the spike domain, several authors have proposed to convert ANNs trained with the well-established backpropagation algorithm to SNNs so that the latter can be used as inference engines [62,91,92]. ANN-to-SNN conversion imposes that the firing rate of a spiking neuron in the SNN be proportional to the activation output of a non-spiking neuron in the ANN. Various techniques such as weight normalization, noise addition, lateral inhibition or spiking rate based pooling masks, which is similar to max pooling operation, have been employed to this end. Note that this approach is not suitable for implementing systems that can learn in hardware and in real-time.

There are also several efforts directed towards developing architectures with adaptive and evolving network structures [93–97]. SpikeTemp and SpikeComp are algorithms where neurons are progressively added in the classifier layer as the training algorithm approaches the optimal point [95, 96]. An SNN with evolving architecture called NeuCube, directly inspired by the brain [93], incorporates weight adjustments based on supervised and unsupervised rules and additionally, adds new network neurons as per training requirements.

The normalized approximate gradient descent based method (NormAD) for synaptic strength modification in SNNs has been proposed that casts learning as an optimization problem for tuning the membrane potential to create spikes at desired time instants [37]. Compared to the ReSuMe learning rule, at least 10x faster convergence characteristics has been demonstrated using this algorithm for generating arbitrary desired spike streams.

Following subsections discuss the ReSuMe and NormAD learning rules, which have been studied for various network trade-offs for efficient hardware realizations.

2.2.4 Remote Supervised Method (ReSuMe)

The ReSuMe algorithm, developed by Ponulak, [59], adjusts the synaptic weights of the neuron to be trained depending on the desired spikes ($S^d(t)$) as well as based on the spikes observed ($S^o(t)$) in the trainee neuron. It essentially increases the magnitude of the weights at an instant when there is a desired spike and decreases when there is a spike at an undesired time (Figure 2.7). The spike trains $S(t)$ can be represented as sequence of delta (δ) functions as,

$$S(t) = \sum_f \delta(t - t^f) \tag{2.23}$$

where, t^f is the time instant at which a spike occurs. The cost function can be defined as a function of observed and desired spike trains as,

$$\Delta w \propto f(S^d - S^o) \tag{2.24}$$

The ReSuMe learning rule is similar to the STDP rule, in the sense that the weight adjustment depends on precise timing of the input and the output spikes, and additionally the desired spikes. By combining the above proportionality with the STDP (or Hebb's), which incorporates the input (S^{in}) as well, the weight change is given by,

$$\Delta w \propto (S^d - S^o)S^{in}, \tag{2.25}$$

More formally, the weight update, as in Ponulak's dissertation, is given by [59],

$$\frac{d}{dt}w(t) = S^d(t)\left[a_d + \int_0^\infty W^d(s^d).S^{in}(t - s^d)ds^d\right] \tag{2.26}$$

$$+S^o(t)\left[a_o + \int_0^\infty W^o(s^o).S^{in}(t - s^o)ds^o\right] \tag{2.27}$$

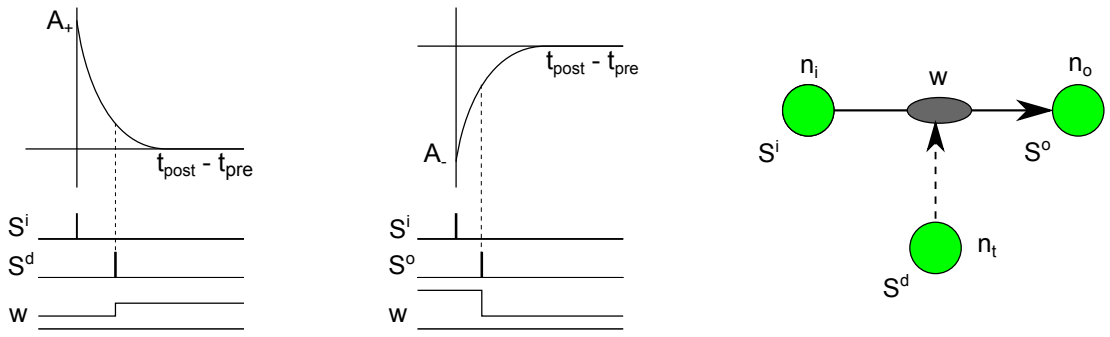

Figure 2.7 Weight update process in ReSuMe follows from the timing principles of the STDP rule, where, the weight update is proportional to the timing difference between the pre- and post-synaptic spike times. However, unlike the unsupervised STDP rule, here, the weight update happens only when there is a spike at the post-synaptic neuron (desired or observed).
Source: [59]

where $W(s)$ the learning window, as in for the STDP rule (Eq. 2.22), is given by,

$$W^d(s^d) = \begin{cases} +A_+^d exp\left(\frac{-s^d}{\tau_+}\right) & if s^d > 0 \\ 0 & if\ s^d \leq 0 \end{cases} \tag{2.28}$$

$$W^o(s^o) = \begin{cases} -A_+^o exp\left(\frac{-s^o}{\tau_-}\right) & if s^o > 0 \\ 0 & if\ s^o \leq 0 \end{cases} \tag{2.29}$$

Figure 2.8 shows a schematic representation of the architecture and training process in ReSuMe. The neural-micro circuit (NMC) block, consists of neurons arranged in a 3D manner, with stochastic connections amongst them and having synaptic delays. The NMC network generates a reservoir of spike, that is applied to the trainee neuron, which is to be trained to generate spikes at specific times. The NMC has a connectivity similar to the connectivity of the neural network in the human brain.

The inputs spike train with Poisson distributed spike times and average spiking rate, $\lambda = 20\,\text{Hz}$, are applied to a fraction of the NMC neurons ($\sim 30\%$), which generates a rich spiking pattern of spikes (as seen in the NMC spike raster). At the output, a fraction of connections from the NMC ($\sim 70\%$) are passed to the output neuron. The connections between the input and the NMC neurons and within

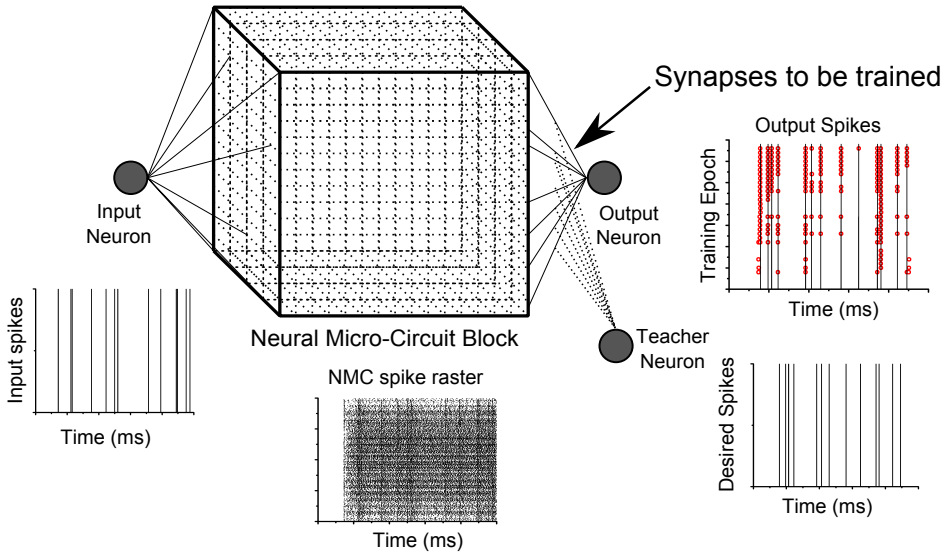

Figure 2.8 ReSuMe network and training process. A single input spike train is applied to an NMC block creating a rich set of spike trains. These spike trains are then applied to the trainee neuron to create the desired set of spikes.

the NMC neurons are stochastic in nature, so as to create a generic architecture. The output spike raster shows the spikes converging to the desired times within an accuracy of 2.5 ms, over different training epochs.

2.2.5 Normalized Approximate Descent (NormAD) Rule

Normalized Approximate Descent rule as discussed in [37], trains the synaptic weights such that the spiking neuron creates spikes at desired time instants. Similar to ReSuMe in the previous subsection, this rule updates the synaptic weights whenever the observed and desired spikes do not match. The weight update follows the standard ANN expression as,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}. \tag{2.30}$$

The weight update term, $\Delta\mathbf{w}$ is calculated only when there is a discrepancy between the spike times in the desired ($S^d(t)$) and observed ($S^o(t)$) spike trains, $e(t) = S^d(t) - S^o(t)$. As described in [37], this is achieved by defining a cost function in terms of the error between the desired ($V_{des}(t)$) and observed ($V(\mathbf{w}, t)$) neuron

membrane potentials as:

$$J(\mathbf{w}) = \frac{1}{2}\int_0^T |e(t)|(V_{des}(t) - V(\mathbf{w},t))^2 dt \tag{2.31}$$

Using gradient descent on the instantaneous cost, the weight update term can be written as:

$$\Delta\mathbf{w}(t) = k(t)\nabla_{\mathbf{w}} J(\mathbf{w},t) \tag{2.32}$$

with

$$\nabla_{\mathbf{w}} J(\mathbf{w},t) = |e(t)|(V_{des}(t) - V(\mathbf{w},t))\nabla_{\mathbf{w}} V(\mathbf{w},t) \tag{2.33}$$

By *normalizing* and *approximating* the dependence of membrane potential on the weights, it is possible to obtain a closed form relationship for the weight update as:

$$\Delta\mathbf{w} = r\int_0^T e(t)\frac{\hat{\mathbf{d}}(t)}{\|\hat{\mathbf{d}}(t)\|}dt \tag{2.34}$$

where,

$$\hat{\mathbf{d}}(t) = \mathbf{c}(t) * \hat{h}(t), \text{with } \hat{h}(t) = \exp(-t/\tau_L)u(t). \tag{2.35}$$

Here, $\mathbf{c}(t)$ is the synaptic kernel (as in Equation 2.14) and $\hat{h}(t)$ is the LIF neuron's impulse response function defined for the period till the neuron spikes. The constant $\tau_L = 1\,\text{ms}$ represents the approximation for the neuronal time constant, during training phase. Normalization helps in eliminating the dependency on $V_{des}(t)$, which is an unknown term. The weight update depends only on the output spike error $e(t)$ and the incoming spike trains, captured in $\hat{d}(t)$. The constant r, having the dimensions of synaptic conductance, is a function of the number of input neurons as discussed in [37]. Figure 2.9 shows the spike raster plot during training. The weight update in NormAD also shows similar trend as in the STDP rule (as presented in Section 2.2). Considering the synaptic weights to be equivalent conductance values

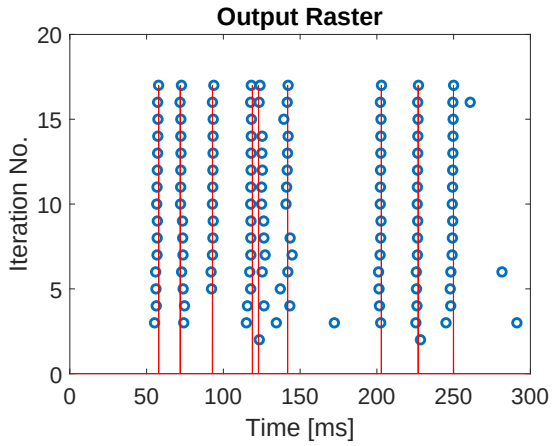

Figure 2.9 Demonstration of NormAD training over successive training iterations (on y-axis). The solid red lines are the desired spikes (with inter-spike times Poisson distributed), while the blue circles represent the trainee neuron's spikes over a period of 300 ms. It can be seen that the algorithm converges in just 17 training iterations.

G, the synaptic or conductance change, ΔG depends on the relative timings of the input and output spikes for a given synapse (see Figure 2.10).

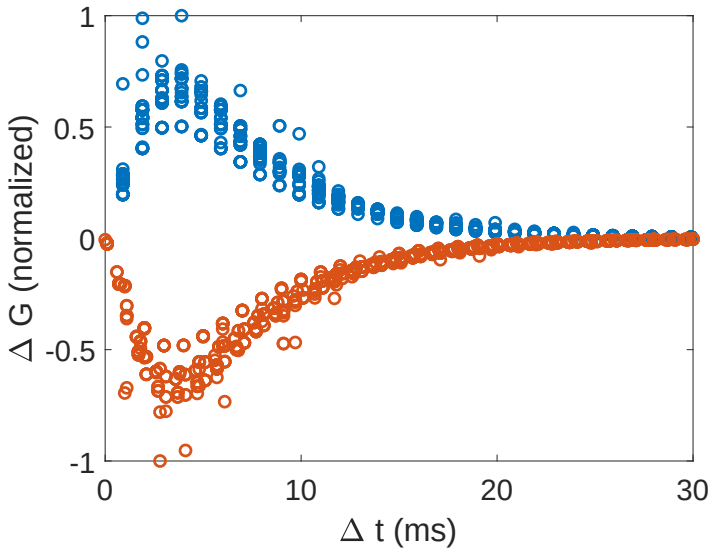

Figure 2.10 While the NormAD is an analytically derived rule, using the neuronal and synaptic dynamics show that the synaptic weight update ΔG has a dependence on Δt similar to biological STDP. Here, $\Delta t = t_o - t_i$, where t_i is the spike time from the input neuron and t_o is the time of spike observed/expected from the output neuron. The curve with blue circles represent synaptic potentiation, when a spike is expected on the output neuron, while the red circles represent the synaptic depression.

2.3 Summary

This chapter presented an introduction to the computationally simple Leaky integrate and fire (LIF) spiking neuron models. We also presented a kernel-based model for the synapse, that perform filtering on the incoming spikes to covert them into input currents for the spiking neurons. Two supervised learning rules for SNN, ReSuMe and NormAD were also discussed, which can be used to train the network to issue spikes at desired instants of time.

In the next chapter, we discuss the application of the NormAD rule to the problem of handwritten digit classification. The different network choices, and the insights from several optimization studies will be explored in the next chapter.

CHAPTER 3

HANDWRITTEN DIGIT RECOGNITION WITH SPIKING NEURAL NETWORKS

This chapter discusses the application of SNNs to the problem of handwritten digit classification. Our work focuses on applying a precise spike based supervised learning algorithm to the MNIST (Modified National Institute of Standards and Technology database) handwritten digit classification problem [49]. The designed SNN employs spiking neurons operating at sparse biological spike rates below 300 Hz and achieves a classification accuracy of 98.17% on the MNIST test database with four times fewer parameters compared to the state-of-the-art. We present several insights from extensive numerical experiments regarding optimization of learning parameters and network configuration to improve its accuracy. We also discuss the prospects of employing precise timing of spikes output from the SNN to make robust predictions on the input class. The methodology described in this chapter is as published in the Neural Networks journal [98]. The later part of this chapter also presents a methodology to accelerate the simulation of SNNs on a GP-GPU platform using the CUDA framework. We also show a real time demonstration of this network to classify users' hand-drawn digits on a touch-pad (non-MNIST images) in real time.

Prior SNN based demonstration of handwritten digit recognition using spiking version of backpropagation of errors has achieved 99.59% accuracy on the MNIST test set using a convolution neural network [90], and another work fom INI Zurich reported 98.7% based on a fully connected 4−layer network and 99.31% with convolutional spiking networks, but all of these had more than 4× higher number of trainable synapses compared to our network [60]. The training algorithm employed in that work has a cost function that is continuous in time defined in terms of the low pass filtered

spike trains (both input and output). Compared to the state-of-the-art networks which have shown over 99% accuracy, our SNN trained with NormAD shows an accuracy of 98.17% on the test set of the MNIST database, with 4× fewer synaptic learning parameters [6,7,13,60]. Furthermore, if the network architecture and number of synaptic parameters are kept the same, we show that the accuracy and performance of the NormAD trained SNN is comparable to that of an equivalent ANN trained using backpropagation, and furthermore, for lower bit-precision representation of the weights, the SNN shows a better accuracy. Very recently, after the work presented in the dissertation was completed, a group from Purdue also published similar results comparing the superiority of SNNs trained with spike based learning rules over other approaches in terms of both accuracy and computations needed [90]. However, their work does not make use of the spike timings in the network output, unlike our work.

Section 3.1 presents the SNN architecture, the input-output encoding and decoding schemes. The network is trained with the NormAD supervised learning rule. Section 3.2 describes several hyper-parameter tuning experiments and the results achieved on the MNIST database. Section 3.3 discusses the optimization of the network for implementation in energy and memory constrained hardware platforms by approximating the neuronal dynamics and using low-precision bits for storing the synaptic weights. The GPU based acceleration of the SNN simulation and the computations needed are discussed in Section 3.4. This section also presents the design of the user interface demonstration, where the SNN running on a laptop GPU is able to infer the digits drawn by users in real time. The details described in this section is also published in [99, 100]. Finally, Section 3.6 summarizes this work and presents a discussion on our results.

3.1 Network Architecture

As illustrated in Figure 3.1, we designed a simple 3-layer SNN for classification of handwritten digits from the MNIST database. Since MNIST images are 28×28 pixels, our network's input layer has 784 neurons and the output layer has 10 neurons, each corresponding to a particular digit. The input layer neurons connect to 8112 hidden layer neurons through twelve *a priori* fixed 3×3 sized convolutional kernels. The synapses connecting this hidden layer to the output layer are trained using the NormAD algorithm.

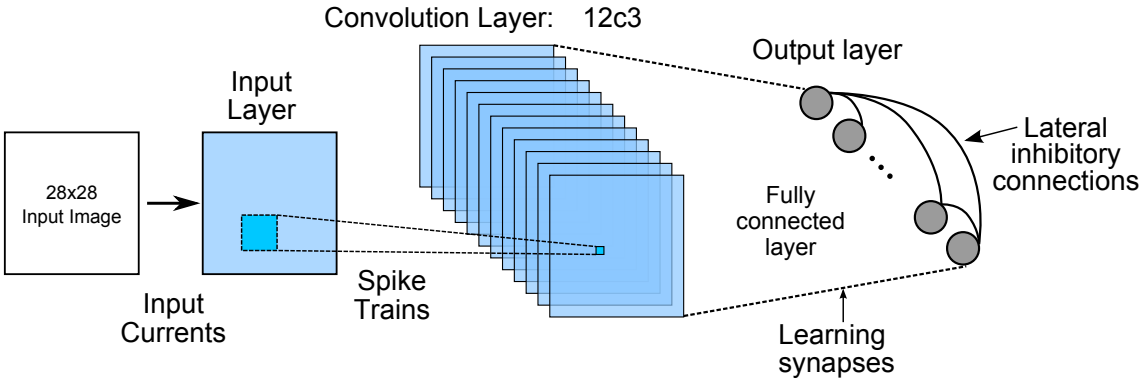

Figure 3.1 The proposed spiking neural network architecture for handwritten digit classification. The spike trains from the input layer with 28×28 neurons are spatially convolved with twelve filters (or convolution kernels) of size 3×3, resulting in the twelve feature maps of size 26×26. The synapses connecting the 8112 convolution layer neurons and the 10 output layer neurons are tuned during training. There is a fixed winner-take-all (WTA) lateral inhibition between the neurons in the output layer.
Source: [98].

3.1.1 Input Encoding

Biological sensory neurons employ complex transformations such as rate coding, time-of-spike coding, population coding and phase coding to encode real-world information in the spike domain [101]. Time-encoding machines that convert band-limited input signals to the spike domain such that their perfect reconstruction is possible have been proposed in [102]. There are also some recent works that use Gaussian receptive fields

or Poisson encoding to directly translate real-valued inputs to spike times [61, 103]. As we are dealing with static images, we translate each gray-scale pixel value, in the range $[0, 255]$, to currents over a certain time-period, that can be applied as inputs to the spiking neurons. Accordingly, each pixel value k is converted into a constant input current for the LIF neuron as:

$$i(k) = I_0 + (k \times I_p)\,. \tag{3.1}$$

Based on our empirical experiments, we use $I_p = 101.2\,\text{pA}$ as a scaling factor and from the neuronal parameters we use $I_0 = 2700\,\text{pA}$ which is the maximum constant amplitude current that does not generate a spike in the LIF neuron in Equation 2.13. As a result, LIF neuron in the input layer issues spikes that are uniformly spaced in time, with a frequency that is sub-linearly proportional to the magnitude of its input current [104].

3.1.2 Convolutional Feature Extraction

The convolution layer of our network uses *a priori* determined fixed weights for the different feature maps and serves to detect the key features of the image. The filter kernels are continuous curves as shown in Figure 3.2(a), and incorporate both excitatory and inhibitory connections. Our kernels are only 3×3 pixels and were inspired by biological studies that suggest that the first few layers of the visual cortex consist of small-sized visual receptive fields [5].

The filter kernels are spatially convolved with 28×28 spike trains arriving from the input layer neurons, over a simulation period T, with a stride of 1, resulting in feature maps of size 26×26. The weight kernels have an overall net higher inhibition than excitation, as it helped to better suppress the spikes from unwanted edges of the input digit image in the corresponding feature map. Fixed weights based on Gabor filters have been used before as the first layer in a deep convolution neural network,

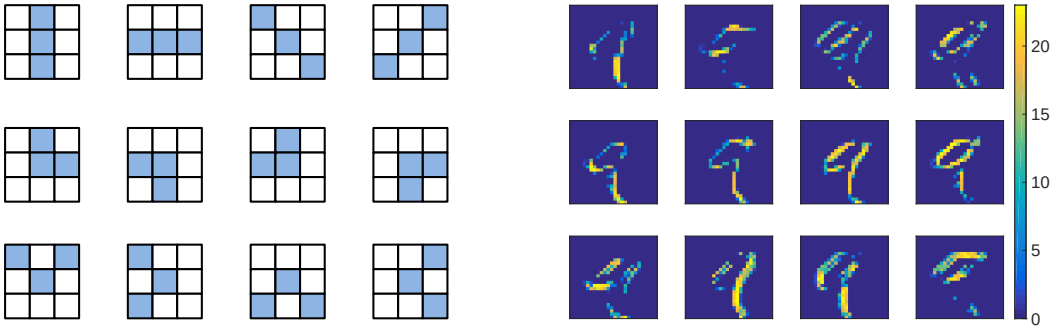

Figure 3.2 (a). (left) Convolution filters used in our SNN are of size 3×3 pixels. The blue pixels are the excitatory weights, while white pixels are inhibitory values. The magnitude of the excitatory weight is 1.6 times that of the inhibitory weight. (b). (right) The twelve spike count feature maps corresponding to these filters obtained when an exemplary image of digit '9' was presented to the network. The color intensities in the 2D map depict the number of spikes generated by the neurons of the hidden layer when the input was presented for $T = 100\,\text{ms}$.
Source: [98].

and have shown an improvement in the accuracy for the MNIST dataset compared to the original LeNet-5 network [6, 105]. We use relatively simpler edge detection filters in the hidden layer of our network.

The spikes from the input layer neurons pass through these synaptic weight kernels to generate currents to the hidden layer neurons. The magnitude of the current entering the hidden layer neurons is scaled such that on an average their output spike rate is limited to 10 Hz. Figure 3.2(b) shows the 2D feature maps depicting the number of spikes generated by the neurons in the hidden layer when an exemplary image of digit 9 from the MNIST data-set is presented to the network for $T = 100\,\text{ms}$. The different kernels are able to effectively encode the edges and features of the input image in spike domain.

3.1.3 Learning Layer

The output layer of the network, where the 8112 neurons from the convolution feature maps connect to the 10 output layer neurons in an all-to-all manner, is trained in a supervised manner. We use the Normalized Approximate Descent Rule (NormAD)

discussed in Chapter 2 to adjust the synaptic weights of this layer. The weight update as discussed previously, is computed as:

$$\Delta \mathbf{w} = r \int_0^T e(t) \frac{\hat{\mathbf{d}}(t)}{\|\hat{\mathbf{d}}(t)\|} dt \tag{3.2}$$

For our network, the learning rate r is set to 200 pS after several empirical studies, where each output neuron to be trained has 8112 synapses.

In our network, the desired signal $S^d(t)$ for the label neuron is a uniform spike train with a frequency of 285 Hz, corresponding to a spike every 3.5 ms, which is slightly higher than the LIF refractory period of 3 ms. There are no spikes in the $S^d(t)$ for all the other neurons.

3.1.4 Lateral Inhibition at the Output Layer

In addition to the feed-forward inputs from the convolution layer neurons, each output layer neuron also receives lateral inhibitory inputs from the remaining 9 output neurons, implementing a winner-take-all (WTA) dynamics, similar to [60]. When a neuron spikes, its outgoing WTA synapses inject a negative current to other neurons, thereby suppressing their spikes, as illustrated in Figure 3.3.

3.2 Hyper-parameter Tuning Experiments

We now discuss the results of various experiments that we conducted in our study to optimize the performance of our network. We start with the baseline experiments that were conducted to analyze network performance, and then discuss the sensitivity of the network to signal encoding parameters such as image presentation duration, learning rate schedules and the network size.

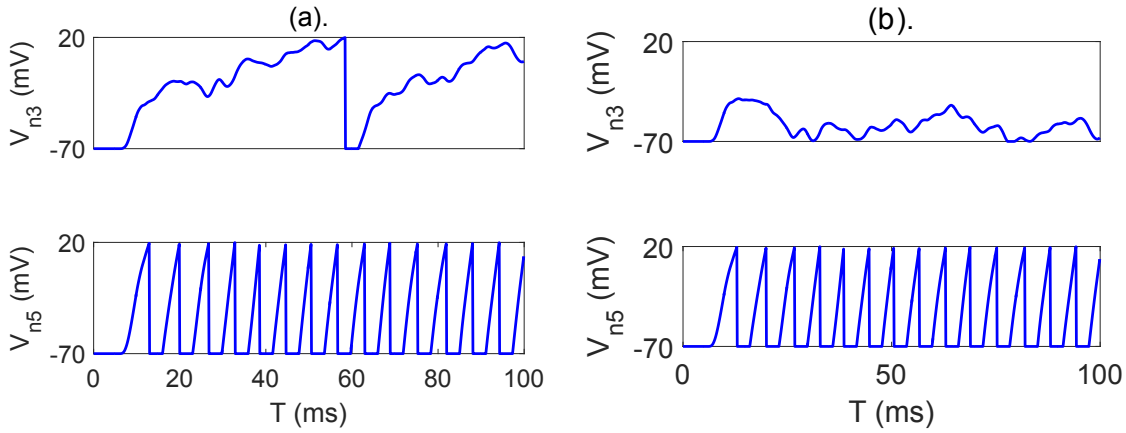

Figure 3.3 Membrane potential of two output layer neurons '3' and '5', when an input image of digit '5' was presented to the network. (a) (left) Membrane potential without lateral inhibition and (b) (right) with lateral inhibition. It can be seen that lateral inhibition has suppressed the incorrect neuron '3' from issuing a spike.
Source: [98].

3.2.1 Training Methodology

During training, each image is presented to the network for a duration T and all the hidden layer weights are updated after every image, similar to a stochastic gradient descent (SGD) rule. We divide the MNIST training set into two parts: $50,000$ for training and remaining $10,000$ for validation. In each training epoch, all the $50,000$ images are presented once to the network. The validation set is used to tune the hyper-parameters of the network such as the variation in the learning rate, optimal number of convolution kernels and the presentation duration as discussed in the following subsections. The network accuracy was determined on the MNIST test set consisting of $10,000$ images.

The dynamics of the SNN is evaluated by numerical integration with a time-step of $\Delta t = 0.1\,\text{ms}$ which is 10 times higher than the learning time constant, $\tau_L = 1\,\text{ms}$ used in the NormAD algorithm (see Section 3.1.3). The network simulation was carried out in a CUDA simulator developed by us, the details of which would be presented in the next section.

3.2.2 Accuracy Metrics in Spike Domain

We primarily used two metrics to measure the accuracy of our network – the first based on the spike count and the second based on the correlation C, of the observed spike trains with respect to a reference spike train. In the count metric, the network's output is decided based on the neuron having the highest spike count. The spike correlation measure [106] between the output spike train $S_i^o(t)$ for each neuron i in the output layer and a reference spike train $S^r(t)$ is defined as:

$$C_i = \frac{\langle L[S_i^o(t)], L[S^r(t)] \rangle}{\|L[S_i^o(t)]\| \, \|L[S^r(t)]\|} \tag{3.3}$$

where

$$L[S(t)] = S(t) * \exp(-t/\tau)u(t). \tag{3.4}$$

Here $\langle \mathbf{x}, \mathbf{y} \rangle$ represents the dot product of vectors $\mathbf{x}$ and $\mathbf{y}$. The training signal with a frequency $f_{out} = 285\,\text{Hz}$ is also used as the reference signal during inference. The neuron with the highest value of C is declared the winner of the classification.

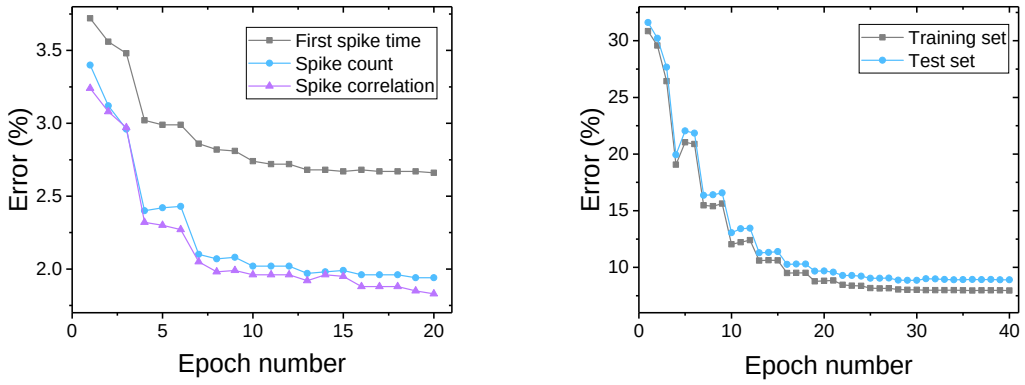

Figure 3.4 (a). (left) The 3-layer SNN error on the MNIST test data-set based on the count, correlation and first-spike-time metrics. It can be seen that the network classification error in terms of first neuron to spike (in gray) during the presentation interval T, is worse by almost 1% compared to either count (blue) or the correlation metric (magenta). (b). (right) For a 2-layer SNN without the hidden layer, the error saturates to about 8%, even at 40 epochs of training, illustrating the importance of the hidden layer.
Source: [98].

The SNN is trained on the MNIST training set for 20 epochs beyond which we did not see any further improvement in the training/validation accuracy. It can be seen from Figure 3.4(a) that precise timing of spikes measured using the correlation metric gives a slightly higher accuracy for classification, though the spike count metric is a simpler metric to evaluate. The classification accuracy of the network is reported using the correlation metric for the succeeding subsections, with explicit mention of the count metric whenever it is used. We also considered the classification accuracy based on the output neuron that spiked first during the input presentation. However, the accuracy based on this metric at the end of 20 epochs was only about 97.34%. While there is a significant drop in accuracy compared to the correlation and spike count metrics, the prediction can be made within 20 ms of image presentation in 99% of input samples using the first-to-spike metric. This trade-off between latency and accuracy may be especially attractive for low-power approximate computing applications.

We also note the crucial role the convolutional hidden layer plays in improving the network accuracy in a 2-layer network with the 784 input neurons connected directly to the 10 output layers, the network's error saturates around 8% (Figure 3.4(b)).

3.2.3 Learning Rate Schedule Optimization

As discussed in [37], the optimal learning rate for the NormAD algorithm depends on the number of input neurons, N_{inp} and scales according to a $N_{inp}^{-1/2}$ rule. We studied several protocols (learning rate schedules) to decrease the learning rate during training (Table 3.1), which resulted in lowering the network error by nearly 0.5% (Figure 3.5).

Epoch dependent learning rate schedules have shown accuracy improvement in previous works for ANN training employing Stochastic Gradient Descent (SGD) [6,26,60]; in our study, we experimented with these and several other schedules, shown

in the Table 3.1. We use the schedule 5 which gave the best validation error after convergence, for the rest of experiments.

Table 3.1 Learning Rate Schedules

Scheme	Learning rate (pS)
Schedule 1	$r_0 = 200$, constant over all epochs, n
Schedule 2	$(1/n)$ decrease: $r(n) = \frac{r_0}{(1+k\times n)}$
Schedule 3	Exponential decrease: $r(n) = r_0 \exp(-k \times n)$
Schedule 4	Step decrease by half every 5 epochs
Schedule 5	Step decrease by half every 3 epochs

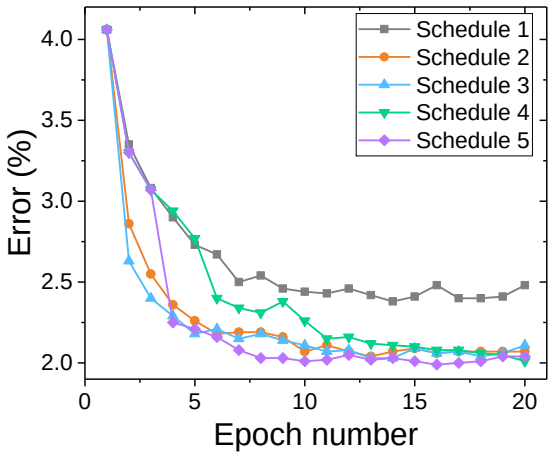

Figure 3.5 Network error on the validation set for five different rate schedules listed in Table 3.1.
Source: [98].

3.2.4 Network Parameter Optimization

We also optimized the design parameters of the network such as the number of the convolution kernels used in the hidden layer and the time period T used for presenting each input image to the network. Larger values of T results in longer integration time to learn the features of each image, as more spikes (or error points) are produced,

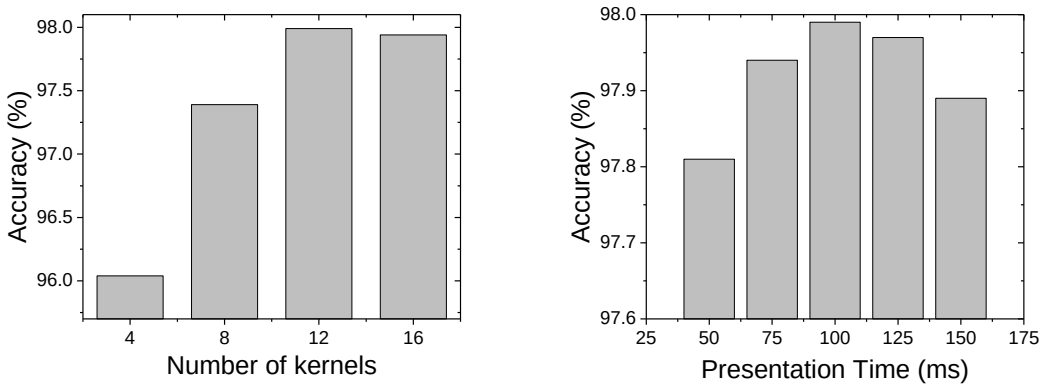

Figure 3.6 (a). (left) Classification accuracy on the MNIST test set as a function of the number of convolutional kernels; (b). (right) the presentation duration, T. The network accuracy is optimized with 12 kernels and a presentation duration of $T = 100\,\text{ms}$.
Source: [98].

resulting in a larger magnitude for the weight update. However, from the perspective of improving the throughput for network performance and preventing over-fitting, smaller values of T are more desirable. Figure 3.6 shows the network performance as a function of the number of convolution kernels and the presentation duration T for the images. The network accuracy is optimized with 12 kernels and a presentation duration of $T = 100\,\text{ms}$. We used a constant inhibitory WTA synaptic strength of $1\,\text{nS}$ for all connections in the output layer.

3.2.5 MNIST Accuracy Results

Having optimized the network hyper-parameters, we trained our SNN with the complete MNIST training dataset ($60,000$ images) for 20 epochs. The SNN achieved an accuracy of 99.82% on the MNIST training set and 98.17% on the test set.

We also trained an equivalent ANN with the same architecture, i.e., the same number of neurons and connectivity patterns (but without the lateral WTA connection) as the SNN in Figure 3.1. We used the rectified linear unit (ReLU) as the activation function of the neurons in this network. The weights of the fully-connected layer were adjusted by the standard gradient descent rule by back-propagating the

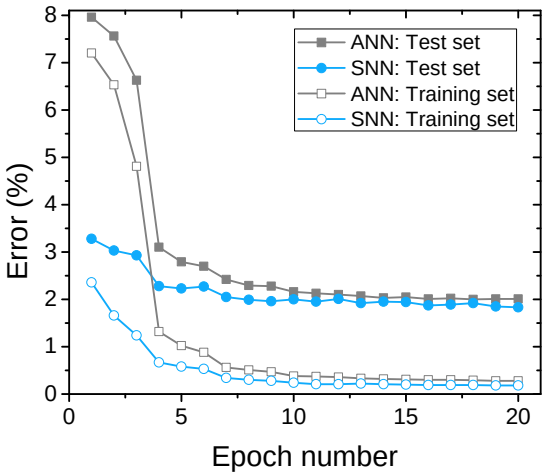

Figure 3.7 Comparison of the MNIST error for the 3-layer SNN and an equivalent ANN with the same network structure during 20 epochs of training. The SNN performance (0.18% error for training set and 1.83% error for test set at convergence) is slightly better than that of the ANN (0.28% error for training set and 2.0% for test set at convergence).
Source: [98].

network error. After fine-tuning the learning rate schedule, this ANN achieved an accuracy of 98.0% on the MNIST test set, which is close to the best case accuracy of around 98.50% reported on an equivalently sized three-layered ANN [107]. The performance for training and test sets for the SNN and ANN networks for 20 epochs of training is shown in Figure 3.7. This comparison shows that SNNs trained using the NormAD algorithm can obtain performance similar to equivalent ANNs in large benchmark classification problems.

Figure 3.8 shows the average of the trained weights of the synapses from the 12 feature maps to each of the 10 output neurons of SNN. When the network is trained on the first 100 images, the weight maps closely resemble the images of the training set digits, though the test set accuracy using these weights was only about 65.8%. When the network is trained with all the $60,000$ images in the training set, the test set accuracy rises to 98.17%, thanks to a more complex representation of the images that are captured by the synaptic weights in the network.

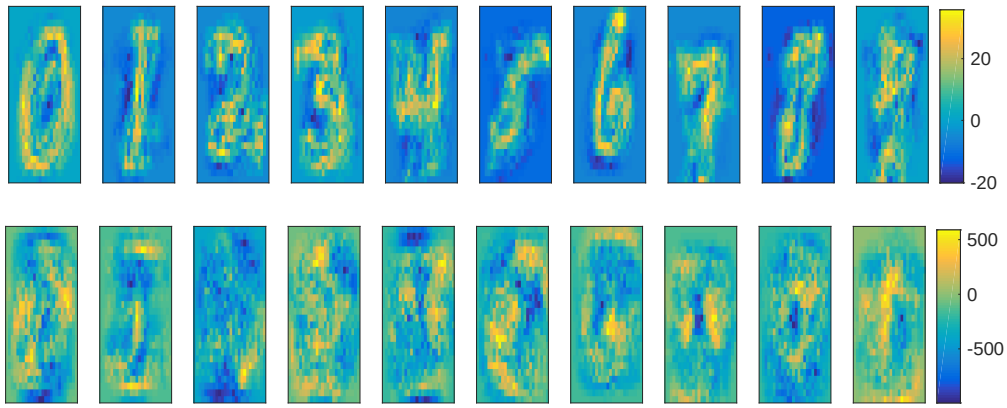

Figure 3.8 Average of the trained weights (in pS) from the 12 kernels in the hidden layer to the 10 neurons in the output layer is the effective internal representation of the digits learned by the network. (Top) The average weights in the output layer of the SNN after 100 images presented once for training (when the test set accuracy was only 65.8%) and; (Bottom) average weights after training (i.e., with 98.17% accuracy). *Source:* [98].

To benchmark the classification performances of our network, we compare the accuracy and number of learning synapses in other state-of-the-art approaches for MNIST handwritten digit classification (Table 3.2). We note that while the accuracy of our approach is about 1.6% smaller than the best in class approach, our network achieves this accuracy with nearly three to twenty times lesser number of trainable synaptic weights.

Table 3.3 presents the confusion matrix for the SNN based classification of the MNIST test data-sets into 10 classes. It can be seen that for all the digits, the true positive rate is 97% and above, demonstrating the high selectivity of the classifier layer, even though not easily discernible from the weight maps (Figure 3.8). Only five images failed to create any spike at the output neurons.

3.3 Network Optimization

We now discuss the network optimization studies to translate the software design for energy and memory constrained hardware platforms. The optimization can be carried

Table 3.2 MNIST Classification Accuracy Comparison

Network and learning algorithm (BP: back-propagation)	Number of learning synapses	Test set Accuracy
ANN (LeNet-5) [6]	331,984	99.05%
GCNN (LeNet-5 + Gabor filters) [105]	331,984	99.32%
MCDNN (Multi-column Deep NN) [7]	1,574,600	99.77%
DNN with DropConnect [108]	2,508,470	99.79%
SNN, with STDP [61]	5,017,600	95.0%
Fully connected SNN, with BP [60]	328,984	98.77%
Convolution SNN with BP [60]	581,520	99.31%
Spiking ConvNet [62]	1,422,848	99.11%
BASNN [46]	268,800	98.17%
Spike based BP [90] (not event-driven)	331,984	99.59%
SNN, with NormAD (this work)	81,120	98.17%
ANN, with BP (this work)	81,120	98.0%

out either by reducing the amount of parameter memory needed or by reducing the number of operations while simulating the SNN.

3.3.1 Low Precision Weight Encoding

The ability of a network to maintain its accuracy even when the precision for storing the network parameters is limited, is crucial for efficient hardware implementations. It has been observed that accuracy degrades significantly when low-precision weights are used for network emulation. For instance, a 5% drop in accuracy (with the MNIST data-set) was observed even with 5-bits of fixed-point precision for the synaptic weights in [109].

Table 3.3 Confusion Matrix for the SNN's Predicted Output

Actual / Predicted	0	1	2	3	4	5	6	7	8	9
0	973	0	3	0	2	2	9	1	4	4
1	0	1126	1	0	0	0	2	4	0	4
2	2	3	1015	4	1	1	0	9	1	1
3	0	2	0	996	0	7	1	1	6	4
4	0	1	2	0	964	0	1	1	5	7
5	0	1	0	6	0	876	3	0	1	3
6	2	1	1	0	5	3	940	0	1	0
7	1	1	6	2	0	1	0	1005	3	7
8	1	0	1	1	1	2	1	3	947	3
9	0	0	2	1	9	0	0	3	6	975
No spike	1	0	1	0	0	0	1	1	0	1
Total	980	1135	1032	1010	982	892	958	1028	974	1009

We test the ability of our SNN and ANN for test set inference with the limited precision of trained weights. We train the weights of both these networks in the double-precision representation and then test it by quantizing the values of these weights, similar to the approach taken in [110] for designing a scalable hardware solution. The histograms of the weights of our SNN and ANN after training with NormAD and gradient descent, respectively, are observed to be log-normally distributed. We observed that dividing the range of weights into linear bins, rather than log-linear bins gives lesser degradation in performance. Figure 3.9 shows the drop in accuracy for our ANN and SNN, as the number of levels for representing the trained weights are reduced. It can be seen that even at 3-bit quantization, the degradation in SNN accuracy is within 1.0% for $T = 100\,\text{ms}$ compared to the floating point baseline. Further, across all quantization values, the degradation in accuracy of

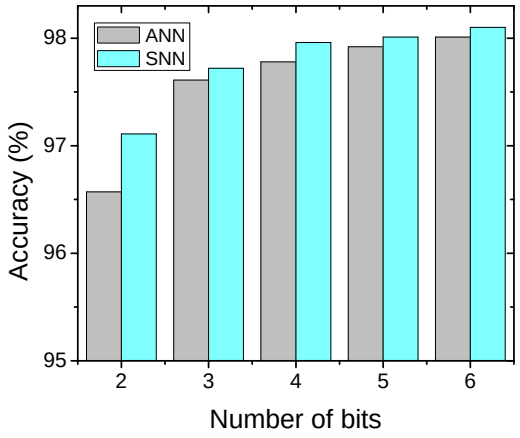

Figure 3.9 Test accuracy as a function of the precision of the trained weights in the SNN and ANN. Even at 2-bit precision, the SNN accuracy is only about 1% lesser than the floating point baseline. Further, the SNN accuracy is better than the corresponding ANN especially at low bit-precision.
Source: [98].

the ANN is slightly worse than that of the spiking network. It is also worth pointing out that compared to previous reports such as [109], where the input spike rate was as high as 1500 Hz, the maximum firing rate in our SNN is restricted to 300 Hz. These results hence, demonstrate the robustness of the SNN architecture and its suitability for memory constrained hardware platforms.

3.3.2 Approximating Neuronal Dynamics

We also study the SNN's performance when the dynamics of the neurons is evaluated with lower precision. As mentioned in the section 3.2.1, the time step for numerical integration was chosen to be 0.1 ms. While this is a requirement for learning, a smaller number of time steps can be used when the network is used for inference, even though there will be some error in the precise time of spike issue.

With $\Delta t = 1$ ms, the number of time-steps needed to compute the response of a neuron are reduced by 10×, thereby speeding up the simulation. Figure 3.10 shows the test accuracy as a function of bit-precision and presentation times for the 3-layer

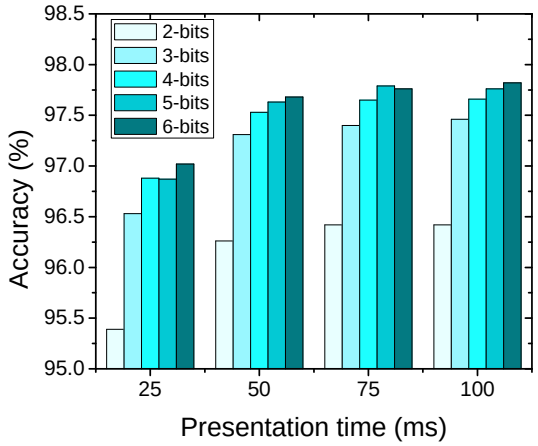

Figure 3.10 MNIST test accuracy (count metric) as a function of bit-precision of weights and the presentation time T, when the neuronal dynamics is approximated with a larger integration time step of 1 ms. Even at 3-bits of precision and with $T = 50$ ms, the drop in accuracy is within 1% of the baseline.
Source: [98].

SNN. Here, we used the count metric to determine the test accuracy to simplify the computation further. Using an integration time-step of 1 ms at a bit-precision of 3-bits, we can infer the class of the digit in just 50 ms or with 50 points of neuronal integration, resulting in an accuracy of 97.31%. Hence, close to base-line accuracies can be maintained in approximate network evaluation that permits higher throughput for classification.

3.4 GPU Implementation of the SNN Training

The SNN is implemented on a GPU platform using the CUDA-C programming framework. A GPU is divided into streaming multiprocessors (SM), each of which consists of stream processors (SP) that are optimized to execute math operations. The CUDA-C programming framework exploits the hardware parallelism of GPUs and launches jobs on the GPU in a grid of blocks each mapped to an SM. The blocks are further divided into multiple threads, each of which is scheduled to run on an SP, also called a CUDA core. Since memory transfer between CPU and GPU local

memory is one of the main bottlenecks, all network variables (i.e., neuron membrane potentials and synaptic currents) are declared in the global GPU memory in our implementation. The simulation Equations (2.13), (2.14) and (2.15) are evaluated numerically in an iterative manner at each time step.

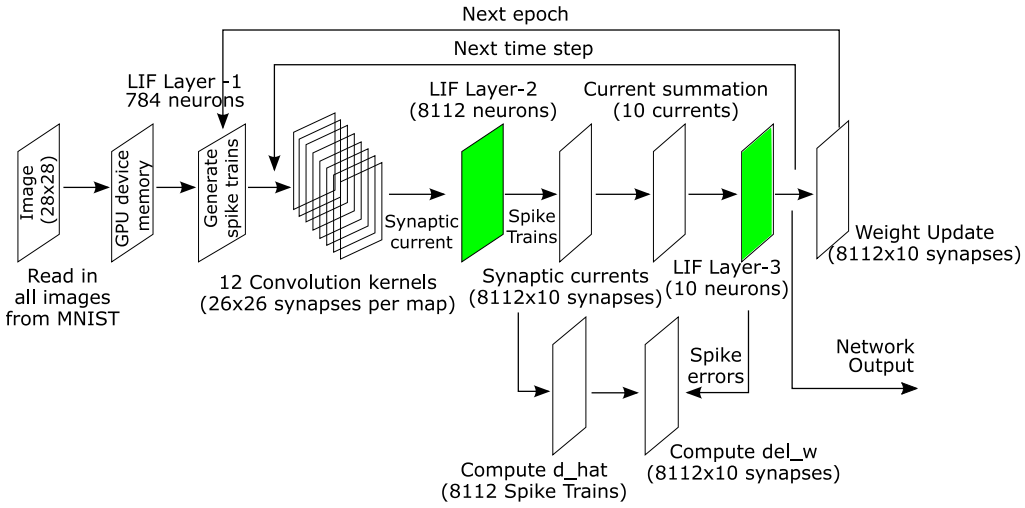

Figure 3.11 Diagram showing the different variables of the network being computed each time step and how the signals flow across different layers. The dimensions within the brackets are the sizes of those variables and their respective CUDA kernels. *Source:* [99].

Figure 3.11 shows the forward pass and backward pass for weight update during the training phase. Image pixels read into the GPU memory are passed as currents to layer one neurons (grid size of 28×28) for the presentation duration, T. The filtering process involves 2D convolution of the incoming spike kernels and the weight matrix (3×3). The computation is parallelized across 12 CUDA kernels, each with a grid size of 26×26 threads. Each thread computes the current to the hidden layer neurons, indexed as a 2D-array i, j, $\{0 \leq i, j, \leq 25\}$ at a time-step n, based on the following spatial convolution relation:

$$I_{in}(i,j,n) = \sum_{a=0}^{2} \sum_{b=0}^{2} w_{conv}(a,b) \times c(i+a, j+b, n) \tag{3.5}$$

where c represents the synaptic kernel (Equation 2.14) calculated from the spike trains of the 28×28 pixels and $w_{conv}(a, b)$ represents each of the weights from the 3×3 filter matrix.

The membrane potential of an array of k LIF neurons, for applied current $\mathbf{I}(n)$ (as described in Equation 2.13) is evaluated using the second order Runge-Kutta method as:

$$\mathbf{k_1} = [-g_L(\mathbf{V_m}(n) - E_L) + \mathbf{I}(n)]/C \tag{3.6}$$

$$\mathbf{k_2} = [-g_L(\mathbf{V_m}(n) + \mathbf{k_1}\Delta t - E_L) + \mathbf{I}(n)]/C \tag{3.7}$$

$$\mathbf{V_m}(n+1) = \mathbf{V_m}(n) + [(\mathbf{k_1} + \mathbf{k_2})\Delta t/2] \tag{3.8}$$

Each thread k independently checks if the membrane potential has exceeded the threshold to artificially reset it.

$$\text{If } V_m^k(n+1) \geq V_T \Rightarrow V_m^k(n+1) = E_L \tag{3.9}$$

Refractory period is implemented by storing the latest spike issue time, n_k^{last} of each neuron in a vector $\mathbf{R}$; the membrane potential of a neuron is updated only when the current time step $n > n_k^{last} + (t_{ref}/\Delta t)$.

The synaptic current from neuron k in hidden layer to neuron l in output layer as given in Equation 2.15 can be re-written to be evaluated in an iterative manner, thereby avoiding the evaluation of expensive exponential of the difference between the current time n and previous spike times n_k^i. The synaptic current computation, at time step n, for each of the (k, l) synapse is spawned in CUDA across 8112×10 kernels as:

$$a_k(n) = a_k(n-1) \times \exp(-\Delta t/\tau_1) + \delta(n - n_k^i) \tag{3.10}$$

$$b_k(n) = b_k(n-1) \times \exp(-\Delta t/\tau_2) + \delta(n - n_k^i) \tag{3.11}$$

$$c_k(n) = a_k(n) - b_k(n) \tag{3.12}$$

$$I_{k,l} = w_{k,l} \times c_k(n) \tag{3.13}$$

where $a_k(n)$ and $b_k(n)$ represent the rising and falling regions of the double exponential synaptic kernel. The strength of the synapses between the hidden and output layers is initialized to zero during training. At every time step, the error function for each output neuron is calculated based on the difference between the observed and desired spikes. Next, $\hat{d}_k$ (Equation 2.35) for the spikes originating from neuron k is computed as:

$$\hat{d}_k(n) = \hat{d}_k(n-1)e^{-\Delta t/\tau_L} + (c_k(n)\Delta t)/C \tag{3.14}$$

Once $\hat{d}_k(n)$ is evaluated, we compute its norm across all k neurons and determine the instantaneous $\Delta w_{k,l}(n)$ for all the $81,120$ synapses in parallel, if there is a spike error. At the end of presentation, the accumulated $\Delta w_{k,l}$ is used to update the synaptic weights in parallel. The evaluation of the total synaptic current and the norm is performed using parallel reduction in CUDA [111]. During the inference or testing phase, we calculate the synaptic currents and membrane potentials of neurons in both layers to determine spike times, but do not evaluate the $\hat{\mathbf{d}}$ and the weight update $\Delta\mathbf{w}$ terms.

3.5 Real-time Inference on User Data

We used the CUDA based SNN described in the previous section, to design a user interface that can capture and identify the images of digits written by users in real-time from a touch-screen interface. The drawing application to capture the digit

drawn by the user is built using OpenCV, an image processing library [112]. The captured image from the touch screen is pre-processed using standard methods similar to that used to generate the MNIST dataset images [49]. We convert the user drawn images to the required format which is a grayscale image of size 28×28 pixels. The network is implemented on the NVIDIA GTX 860M GPU which has 640 CUDA cores and is typically used in laptops. The preprocessing phase takes about 15 ms and this image is then passed to the trained SNN for inference. The CUDA process takes about 300 ms to initialize the network in the GPU memory, after which the network simulation time depends on the presentation time T and the time step interval Δt.

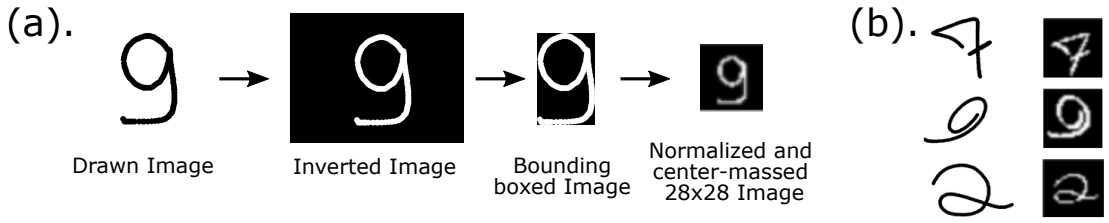

Figure 3.12 (a). Outline of the preprocessing steps used to convert the user input to a 28×28 image that is fed to the network, (b). Examples of user input (left) and the pre-processed 28×28 pixel images fed to the SNN (right).
Source: [99].

3.5.1 Image Preprocessing

Figure 3.12(a) shows the preprocessing steps used to create the input signal to the SNN from the captured image and Figure 3.12(b) shows some sample pre-processed images. The image captured from the user is first binarized by thresholding and cropped to remove excess background. The image is resized to 20 pixels along its longer dimension, while maintaining its aspect ratio. Thereafter, the resized image is placed in a 28×28 bounding box such that the image's center of mass coincides with the center of the bounding box. Finally, the image is passed through a blurring filter to create gray-scale images similar to the ones in the MNIST dataset.

3.5.2 Real-time Simulator

We used the network trained on the MNIST data-set which achieved an error of 0.2% on the training set and 1.94% on the test set. The maximum spike count in the output layer of the network was used as the decision making metric. The network was simulated with a time-step of $\Delta t = 0.1$ ms for $T = 100$ ms.

As we saw in section 3.3, that if the integration time step interval used during inference is 1 ms (i.e., approximating the neuronal integration) instead of 0.1 ms, the MNIST test error increases only by about 0.4% (see Figure 3.13(a)), but there is a 10× reduction in the processing time. Hence, for our touch screen based interface system we simulate the SNN with Δt of 1 ms to infer the users' digits. When each digit is presented for $T = 75$ ms, the network can be simulated in an average wall clock time of 65 ms, making real-time processing possible (Figure 3.13(b)). We tested

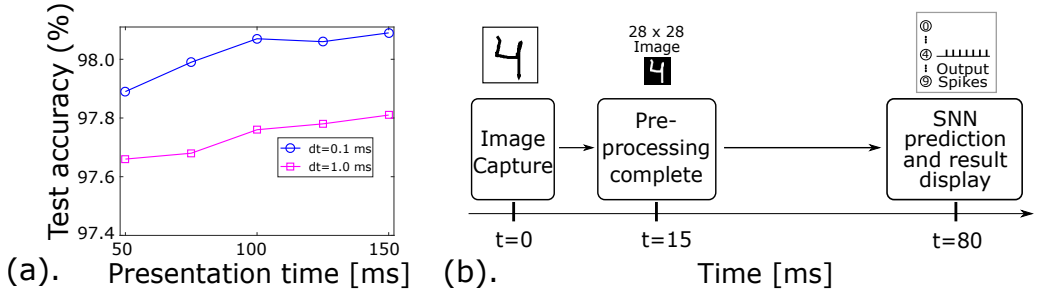

Figure 3.13 (a). MNIST test-set accuracy as a function of presentation time and the integration time step Δt. (b) Various stages of classifying a user's input: the image pre-processing takes 15 ms and the 75 ms SNN emulation is completed in real-time. *Source:* [99].

the network's accuracy with $\Delta t = 1$ ms on a set of 500 handwritten digits collected from various users through our user-interface system. At $T = 75$ ms, we measure an accuracy of 97.4% on our set of 500 captured images, while on the MNIST test-set it was 97.68%. The slight loss in performance compared to the MNIST dataset is attributed to the deviations from the statistical characteristics of the captured images compared to the MNIST dataset.

3.6 Summary

We presented a highly compact and efficient 3-layer spiking neural network for identifying handwritten digits, that achieved an accuracy of 98.17% on the MNIST data set using the NormAD learning algorithm. All information in the network is encoded and processed in the spike domain at sparse biological spike rates. Our studies show that using the precise time of spike issue for classification gives slightly better accuracy compared to the simpler rate coding method. We have also presented two techniques to co-optimize the network for hardware implementation, by reducing the bit-precision of weights and approximating the neuronal dynamics with higher integration time-step size.

The best convolution networks in both spiking and non-spiking versions that have achieved over 99% accuracy on the MNIST database use at least over $300,000$ adjustable synapses. The NormAD-trained SNN, on the other hand, has $4\times$ fewer learning parameters, making it amenable for implementation on custom neuromorphic hardware with on-chip learning. Our studies also show that as low as 3-bits of weight precision is sufficient to maintain close to baseline accuracies in the SNN when used for inference. Compared to an equivalent ANN with similar network architecture, the spike based training approach also shows better accuracy, especially at lower precision for synaptic weight storage. In conclusion, we show that SNN based learning and inference engines are ideally suited for efficient implementation in energy and memory constrained hardware platforms.

We have also demonstrated a general framework for implementing spike based neural networks and supervised learning with event-triggered weight update rules on a GPU platform. At each time step, the neuronal spike transmission, synaptic current computation and weight update calculation for the network are all executed in parallel in this framework. Using this GPU implementation, we demonstrated a touch-screen based platform for real-time classification of user-generated images. The

trained network implemented on the CUDA parallel computing platform is also able to successfully identify digits written by users in real-time, demonstrating its true generalization capability.

CHAPTER 4

COMPACT MODELS FOR NON-VOLATILE MEMORY DEVICES

The past decade has seen an increasing growth in the research efforts for developing nanoscale non-volatile memory (NVM) devices [113]. Emerging NVM technologies such as PCMs (Phase Change Memory), STT-RAMs (Spin Transfer Torque RAM), and RRAMs (Resistive RAM) with a smaller footprint than CMOS-based SRAMs (Static Random Access Memory) have shown great potential to replace existing memory technologies. Moreover, there have been several demonstrations of these NVM devices in designing efficient neural network hardware accelerators, which make use of their analog resistance levels to store the network parameters. Such architectures do not suffer from the von Neumann memory-processor bottleneck and have the potential to realize area and power efficient designs [26, 31, 114]. In this chapter we discuss the basic device physics of these three NVM devices. We then discuss the development of compact and mathematically well-posed models in Verilog-A which we designed for these technologies for faster architecture design and analysis.

While PCM, STT-RAM, and RRAM have shown tremendous potential to replace the existing SRAM and DRAM technologies, there are several reliability related challenges associated with these devices that need to be considered while using them in hardware designs, especially in large arrays with high integration density [115]. Reliability is also an important consideration while realizing neural network algorithms on memristive arrays with multi-bit storage per device [116]. Typically, faults can be either soft, which can be corrected with programming, or hard, wherein the device is permanently stuck at high resistance state (HRS) or low resistance state (LRS). These issues manifest in devices as read disturbance due to

the programming of neighboring cells, hard faults where the device is permanently in high or low resistance states irrespective of programming voltage applied across it and the drift in resistance values over time [117–120]. It has also been reported that hard faults, or stuck-at faults occur in at least 10% of the devices in a fabricated chip [121]. The process of designing and analyzing such architectures requires the use of compact models for these nanoscale devices, which is important while designing peripheral read and write circuits, and performing the subsequent analysis of the whole system.

In this chapter, we present our work on developing compact models for PCM, STT-RAM, and RRAM devices in Verilog-A. Each of these models capture the high-level switching dynamics and are mathematically well-posed and support the three simulation modes - transient, DC, and AC [122, 123]. We also present schemes for adding the reliability aspects of conductance variabilities and stuck-at faults to each of the compact models discussed here. The details of the well posed PCM model discussed in this chapter is as published in [124], and the details of other two models including modeling the reliability aspects of all three devices are as published in [125].

The remainder of this chapter is organized as follows. Section 4.1 presents the compact model for PCM, the process of creating a resistance distribution and adding hard faults into the model. Section 4.2 discusses the basic well-posed STT-RAM compact model. It presents the process of modeling the stochastic switching, conductance variability and hard-faults in the model. Section 4.3 discusses RRAM device model and its reliability aspects. Finally, Section 4.4 summarizes this chapter and presents the the scopes for future research in this area.

4.1 Phase Change Memory

Phase change memory (PCM) is a non-volatile memory technology that is emerging as a leading contender for storage class memories as well as in-memory computing

applications based on crossbar array architectures with co-located memory and processing units [126–128].

For designing neuromorphic and in-memory computing architectures using non-volatile memory (NVM) devices, efficient compact models are required, capturing their key operating characteristics. We present a well-posed model for phase change memory (PCM) devices based on a $Ge_2Sb_2Te_5$ (GST) chalcogenide material. Several models for the GST based PCM device have been proposed earlier, capturing various aspects of the device physics [129–139]. The hierarchical model in [129] incorporates the device geometry and device physics, though it is computationally quite expensive. Simpler models have also be developed using basic circuit elements such as controlled voltage/current sources, resistances and capacitances to model the device behavior [130, 131]. The model presented in [130] captures the binary switching of the device from crystalline to amorphous state, and vice versa. The state switching is based on the analysis of the quenching time once the device temperature exceeds the melting point. A further enhancement of this model showed the capability of demonstrating multiple resistance states within the device as a function of programming current and quenching time [131]. A single equation was used to define the model's I-V response, describing both the off state (low field) and the on state (high field) behavior, with a set of blending functions used to combine the two regimes.

Our model, based on the one discussed in [131], is developed in Verilog-A. It is computationally simple as it uses basic circuit elements and successfully captures the high level dynamics of resistance switching, including its dependence on programming voltages, currents and pulse time-scales. Our results also show the capability of the model to emulate multi-bit storage characteristics of PCM, with pulse-width and pulse-quench based programming schemes. In addition, our model is well-posed and supports different modes of simulation including transient, DC and AC, as per the

guidelines discussed in [122, 123], which so far has not been addressed in previous models.

4.1.1 Device Physics and Operation

Phase change memory (PCM) device consists of a chalcogenide alloy sandwiched between two metal electrodes. Depending on the amplitude and timescales of the programming pulses, the chalcogenide material can be heated and quenched to a high resistive amorphous phase (also called the RESET state) or re-crystallized to a low resistive poly-crystalline phase (also called the SET state) [126–128]. This transition between the crystalline and amorphous regions through Joule heating is reversible and allows the device to be used as a programmable resistor, and hence as a memory storage device. Typically observed contrast in conductance (ON/OFF ratio) of PCM devices is between $100 - 1000$. Furthermore, by controlling the amplitude and falling rate of the programming pulse, it is also possible to gradually change the shape of the amorphous volume in the critical current path of the device, enabling stable analog changes in conductance, and multi-bit storage [128, 140].

Different structures for the PCM devices have been demonstrated such as the mushroom, pore and bridge cell configurations [126]. The mushroom cell is more widely used, where the bottom electrode, which also acts as the heater, is much smaller in diameter ($< 50\,\text{nm}$) compared to the top electrode dimension [126]. More recently, confined PCM cells have been demonstrated, which enables strong control of Joule heating even in the presence of voids [141, 142].

4.1.2 Compact Model

Building up on a previously published discontinuous model [131], we have developed a well-posed Verilog-A model of the device (Figure 4.1), where the device resistance is calculated based on a lumped parameter c_x that represents the crystalline fraction

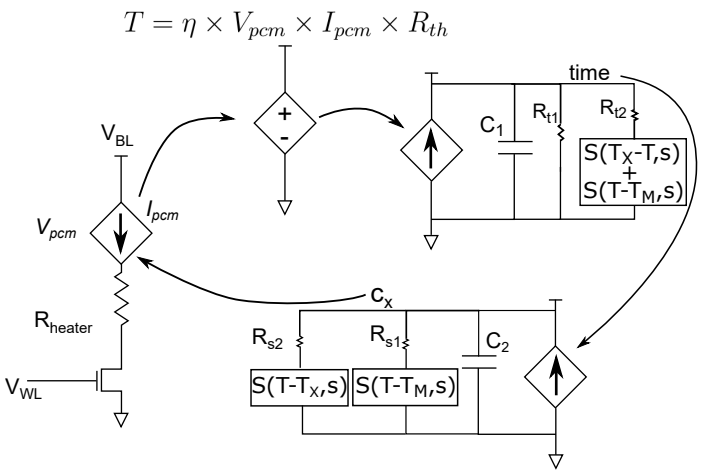

Figure 4.1 Schematic of the continuous compact Verilog-A model for the PCM device. This model is inspired from the one in Ventrice, *et al.*, 2007, and also satisfies the guidelines for compact model development specified in Wang, *et al.* 2016. *Source:* [122, 124, 131].

within the device. The current through the device is given as:

$$I_{PCM} = I_\alpha(V_{PCM}, c_x) + F(I_{PCM}; I_{th}).[-I_\alpha(V1, c_x) + F(V_{PCM}; V_{hold}).I_{ON}(V_{PCM})] \tag{4.1}$$

where the OFF state current I_α and programming region current I_{ON} is:

$$I_\alpha = \frac{\exp(n(c_x).V_{PCM}) - 1}{n(c_x).R(c_x).\exp\left(\frac{E_a(c_x)}{k_B}\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)\right)} \tag{4.2}$$

$$I_{ON} = \frac{V_{PCM} - V_{hold}}{R_{ON}} \tag{4.3}$$

I_{th} represents the threshold current below which switching does not take place and V_h is the hold voltage, which is the x-intercept of the linear region of the I-V curve (see Figure 4.2). I_{ON} represents the current during the ON state or programming state of the device, when the conductance of the device increases. Note that the device resistance R_{ON}, when sufficiently large currents are passing through the device (such that some portion of the chalcogenide alloy is in the molten phase), is significantly lesser that the typical SET resistance of the device.

Based on the device current and voltage, the temperature (T) rise in the material is estimated using a lumped thermal resistance model, assuming a heating efficiency

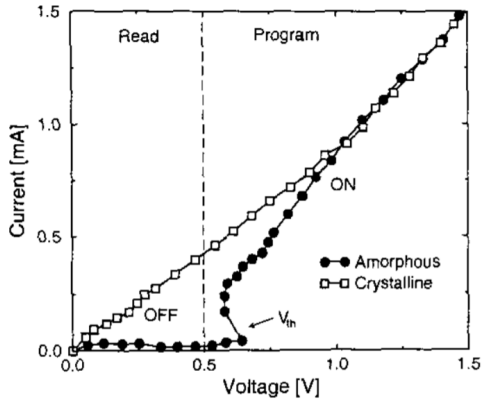

Figure 4.2 Typical experimental I-V characteristics of PCM device (reproduced from Pirovano, *et al.*, 2002).
Source: [143].

of $\eta = 1\%$ [144]. We also assume that in scaled PCM devices, the thermal time constant is significantly smaller than typical SET and RESET pulse-widths.

The crystallization dynamics in the device is described using the Johnson-Mehl-Avrami-Kolmogorov (JMAK) expression [145], giving the instantaneous crystallization fraction c_x within the device, which is then used to approximate the device resistance using the following expressions:

$$c_x = 1 - \exp\left(-t/\tau(T)\right) \tag{4.4}$$

$$R = R_{SET} \times c_x + R_{RESET} \times (1 - c_x) \tag{4.5}$$

where $\tau(T) = A\exp(\Gamma_a/k_B T)$ and the device resistance varies between the SET ($R_{SET} = 10\,\mathrm{k}\Omega$) and RESET resistance ($R_{RESET} = 1\,\mathrm{M}\Omega$). The functions F in Equation 4.1 are the blending functions that combine the different regions of the model's I-V curve. We used a smooth step function $S(x, s)$ to model temperature controlled switches in the auxiliary circuits (Figure 4.1) [122]. Here, x is the controlling signal and s decides the smoothness of the function. These blending

functions are given by the following expressions:

$$F(x; x_{th}) = \frac{1}{(1 + \exp(-(x - x_{th})/\alpha))} \tag{4.6}$$

$$S(x, s) = 0.5\left(\frac{x}{\sqrt{x^2 + s}} + 1\right) \tag{4.7}$$

where, s is the smoothening parameter controlling the transition rate of the step function $S(x, s)$. The different parameters used in our model are listed in the Appendix B in Table B.1.

We follow the guidelines listed by the NEEDS (Nano-Engineered Electronic Device Simulation) initiative [123] to develop a mathematically well-posed model that converges for different modes of simulations such as transient, AC and DC. For this to be satisfied, the Verilog-A model should adhere to a set of coding guidelines. Some of the key requirements include describing the model as a set of Differential Algebraic Equations (DAE) and avoiding any abrupt bias dependent switching statements in the Verilog-A code. The I-V expression for the PCM (Equation 4.1) as presented in [131] is an algebraic expression. By employing smoothened step functions (Equation 4.7), we eliminate sharp temperature dependent switching in the auxiliary circuit (which computes the quench time and crystalline fraction) of the model. Hence, our continuous model is capable of operating in all the different modes of simulation.

4.1.3 Simulation Results

We simulated the switching behavior of the PCM device connected in series with an NMOS access transistor from a commercial 65 nm CMOS technology. Our simulations show that the model captures the essential features of typical I-V characteristics (Figure 4.3) similar to what is seen in the experimental device data in Figure 4.2. The simulations were conducted by applying a voltage ramp at the Word Line (WL) for the two states of initialization (SET and RESET).

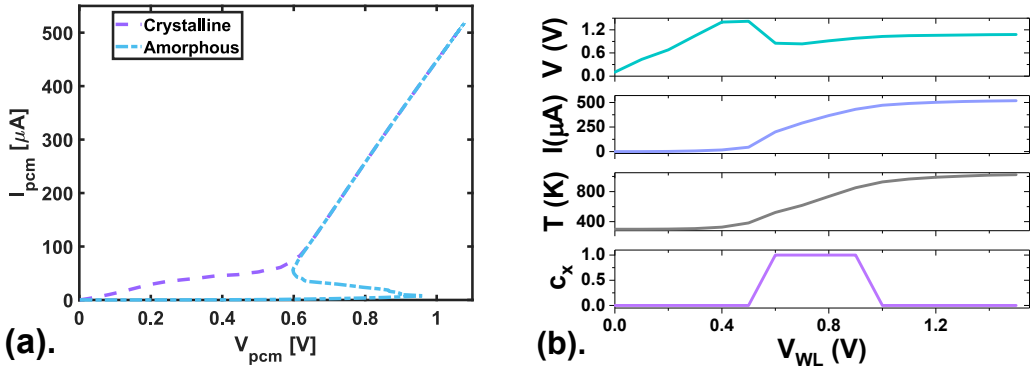

Figure 4.3 (a). I-V response of the well-posed PCM Verilog-A model. The behavior was captured by applying a voltage ramp at the Word Line (WL), i.e., the gate of the access NMOS for the two states of initialization (SET and RESET). (b). DC behavior of the PCM device from HSPICE simulations. From top to bottom: the waveforms of the PCM voltage drop (V), current (I), temperature (T) and the crystalline fraction (c_x) as a function of V_{WL}.
Source: [124].

The model also works in the DC mode; Figure 4.3(b) shows the steady state response of the internal parameters, T and c_x at different voltage values applied to the gate of the access transistor (V_{WL}).

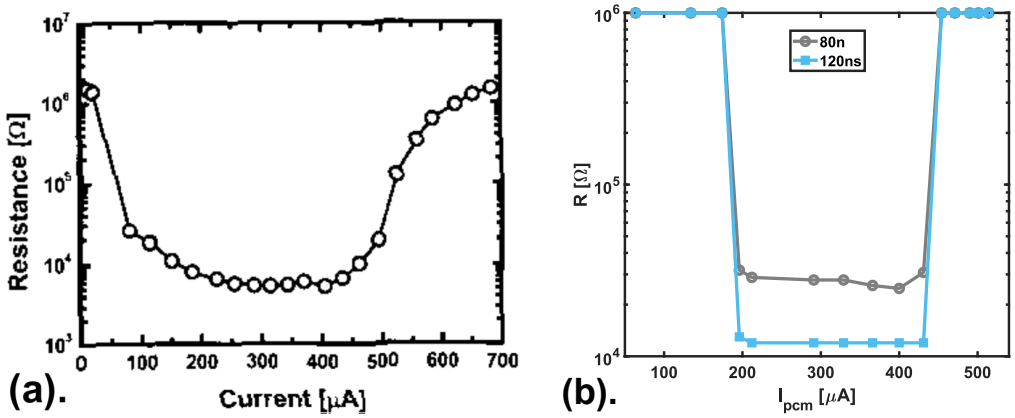

Figure 4.4 (a). Experimental resistance vs. programming current characteristics of a PCM device (reproduced from Pellizzer, *et al.*, 2004). (b). The R-I curve obtained using the model for programming pulse widths of 80 ns and 120 ns.
Source: [124, 146].

Next, we compared the dependence of the device resistance as a function of the applied programming current. Typical experimental characteristics of a PCM

device is shown in Figure 4.4. Starting from a high resistance state, the device resistance can be reduced by applying low amplitude SET pulses; however, once the applied pulse amplitude is sufficiently large to induce melting in the chalcogenide, the device is programmed to its RESET state. The proposed model's behavior shown in Figure 4.4(b) closely matches this experimentally observed programming trend.

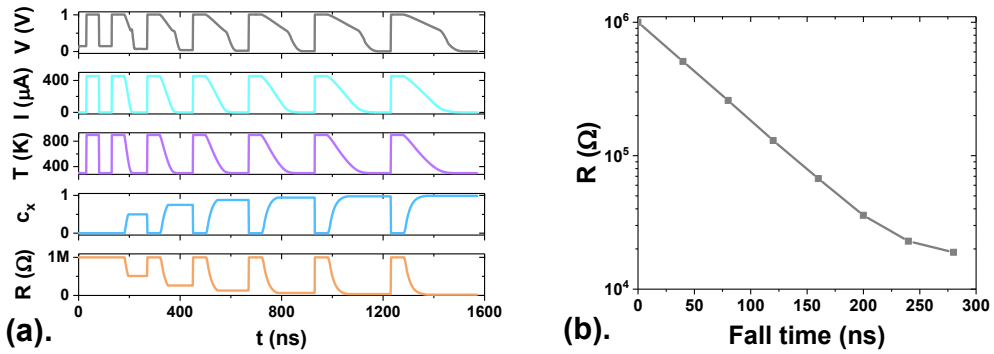

Figure 4.5 (a). Simulated device parameters as a function of duration of the falling edge of the input pulse. Application of each pulse initially resets the device and then brings the final resistance to an intermediate value below R_{reset} depending on the duration of the falling edge. From top to bottom: the waveforms of the PCM voltage drop (V), current (I), device temperature (T), crystalline fraction (c_x) and the device resistance (R). (b). Final resistance of the PCM device as a function of falling edge of input pulse, plotted from the resistance values as shown in Figure 4.5(a).
Source: [124].

We also studied the response of the device model to show multiple conductance levels between R_{SET} and R_{RESET} using two approaches. Dependence of the device resistance on quench time has been demonstrated in [147]. We also employed the same scheme in the simulations by applying pulses having variable fall times in the range of 50 ns to 300 ns. The last two panels in Figure 4.5 show the gradual change in the crystalline fraction and the device resistance as the fall time of the programming pulse is increased. Figure 4.5(b) shows the programmed resistance from the model as a function of the fall time.

Another method of programming the device to various intermediate conductance levels by applying partial SET pulses was demonstrated in [148]. We also simulated the similar behavior using the model, by applying a sequence of short duration

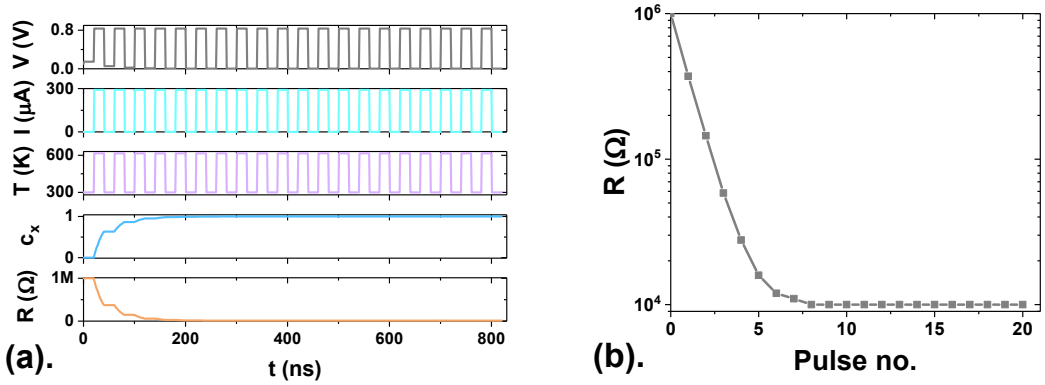

Figure 4.6 (a). Gradual conductance change by application of a series of low amplitude partial SET pulses. From top to bottom: the waveforms of the PCM voltage drop (V), current (I), device temperature (T), crystalline fraction (c_x) and the device resistance (R). (b). PCM resistance as a function of the number of partial SET pulses, plotted from the resistance values as shown in (a).
Source: [124].

pulses, as shown in the waveforms in Figure 4.6(a). Each programming pulse has a constant amplitude ($V_{WL} = 0.7\,\text{V}$) and duration (20 ns). Figure 4.6(b) shows the device resistance as a function of the number of programming pulses.

4.1.4 Reliability in PCM Devices

While the NVM devices of PCM, STT-RAM and RRAM have shown tremendous potential to replace the existing SRAM and DRAM technologies, there are several reliability related challenges associated with these devices that need to be considered while using them in hardware designs [115]. Reliability of these devices is an important factor to be considered when designing large arrays with high density integration. These issues manifest in devices as read disturbance due to the programming of neighboring cells, hard faults such as device permanently being in high or low resistance states irrespective of programming voltage applied across it and the drift in resistance values over time [117–120].

In a deterministic case, for every programming pulse amplitude results in a unique resistance level. However, in practical cases, the programmed resistance state

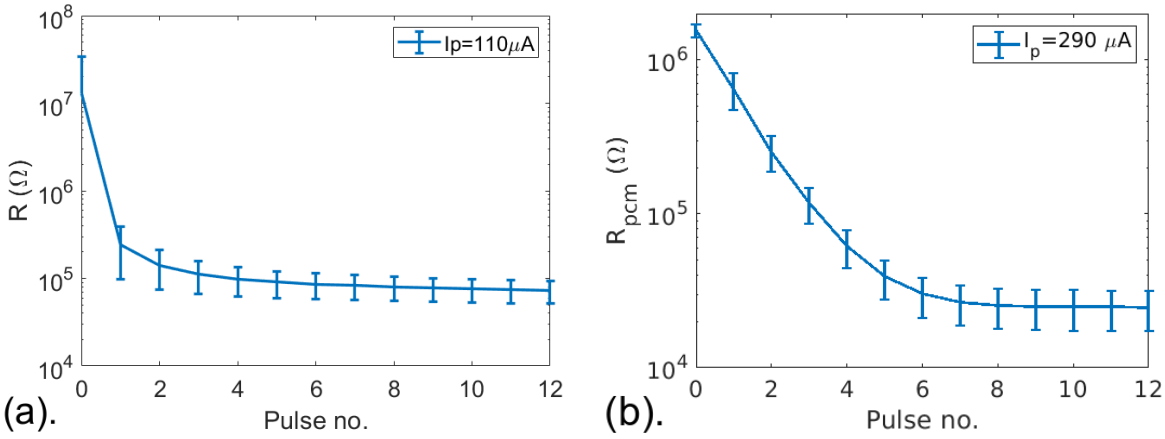

Figure 4.7 (a). Resistance distribution in the IBM's PCM device for programming current of $I_p = 110\,\mu$A and pulse width of 50 ns long programming pulses, as reported in Nandakumar, *et al.,* 2017. (b). PCM model showing intermediate resistance states between the high and low resistance state, when applied with partial SET programming pulses starting from a RESET state. The programming pulses were applied to 100 instances with $I_p = 290\,\mu$A and pulse width of 20 ns.
Source: [148].

shows a distribution around the mean value across multiple devices [148]. We model this phenomenon by adding a noise term to each of the RESET and SET resistance values in Equation 4.5 as:

$$R = (R_{SET} + n_{SET}) \times c_x + (R_{RESET} + n_{RESET}) \times (1 - c_x) \tag{4.8}$$

where n_{SET} and n_{RESET} are the additive noise terms for introducing variabilities at the SET and RESET resistance values, respectively.

Figure 4.7(a) shows a distribution of resistance values from an IBM PCM device, at different partial SET pulses from [148]. Figure 4.7(b) shows the distribution obtained by the simulation of the model using Equation 4.8.

Stuck-at faults (also known as hard-faults), wherein the device remains at either at SET or RESET occur in PCM devices due to field induced migration of ions leading to stuck-SET failures or the device being open circuited after a large number of programming cycles (stuck-RESET failures) [117, 119]. Figures 4.8 (a) and (b) show the model changes and the corresponding simulation waveforms for stuck-set faults.

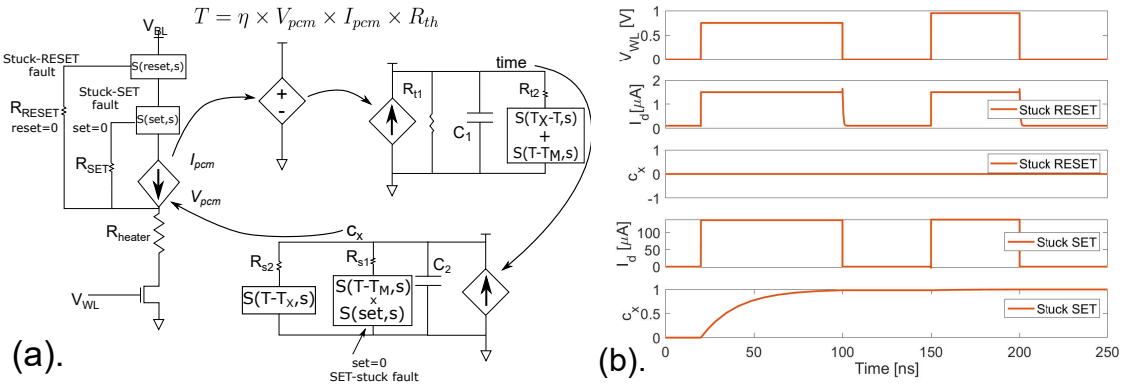

Figure 4.8 (a). PCM model modifications showing the incorporation of stuck-set and stuck-reset faults. The fault variables *set* and *reset* are passed from the SPICE netlist. (b). HSPICE simulation waveforms showing the stuck-SET and stuck-RESET schemes. The first voltage pulse (top panel) is a SET programming pulse followed by a RESET programming pulse. It can be seen in the second and third panels that the current remains low and crystalline fraction at 0 indicating the device is stuck-at RESET state. The last two panels show the case for stuck-SET fault.

When the model instance shows either of these faults, it behaves as a simple resistor with value $R = R_{RESET}$ or $R = R_{SET}$, with the model's VCCS being bypassed. The well-posed PCM model is designed in such a way that the initial value of the crystalline fraction variable as determined by the dc analysis is 0. Hence, in Figure 4.8(b) once the device is programmed to SET state, it does not change its state when the RESET pulse is applied. The PCM model, while being instantiated in a array level netlist,

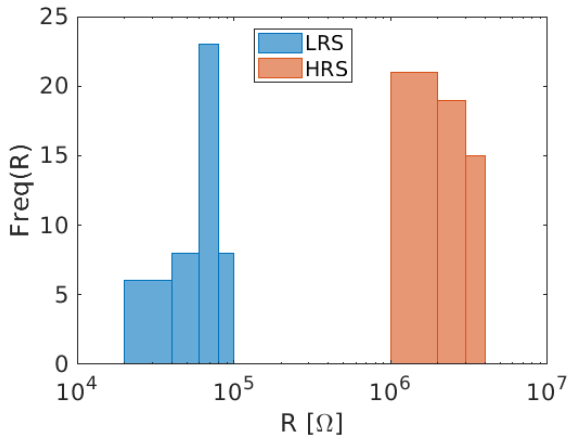

Figure 4.9 Device resistances in an array of 100 devices, with 60% showing stuck-at RESET fault. Similar scheme can be applied to simulate stuck-at SET faults or both versions of faults in a large array of PCM devices.

takes in the parameters which indicate if the device is in Stuck-SET or Stuck-RESET state and the probability of the devices in an array being at either of the two fault states. Figure 4.9 shows the probability distribution of the resistances in an array when devices show a 60% probability of being in stuck-SET or stuck-RESET.

The initialization of the control signals for mimicking stuck-at '1' or stuck-at '0' faults in the model instance are performed in the *initial* block of the Verilog-A code. This is a generic scheme which we follow for all the three device models.

4.2 Spin Transfer Torque RAM

Spin Transfer Torque Magnetic RAM (STT-MRAM) is a two terminal memory device that can store information magnetically but can be read and written electrically. It consists of a Magnetic Tunnel Junction (MTJ), where two ferromagnetic layers are separated by an insulating barrier (typically MgO). The magnetization ($\vec{M}$) of one layer, called the pinned layer (PL) is fixed and aligned in a particular direction called the easy axis, while the magnetization of the second layer called the free layer (FL), is free to rotate and its orientation decides the overall resistance of the MTJ [149]. Compared to other nano-scale NVM devices, STT-RAM has a much higher endurance and switching speed, making it suitable for designing high throughput accelerators [150]. The resistance of the device is high (R_{AP}) when the relative directions of magnetization in the two layers are aligned anti-parallel to each other, and low (R_P) when the two directions are parallel.

4.2.1 STT-RAM: Device Physics

The resistance of the STT-RAM device is high (R_{AP}) when the relative magnetization directions of the two layers are aligned anti-parallel to each other, and low (R_P) when the two magnetizations are parallel. The resistance change is denoted by a term called the Tunnel Magnetic Ratio (TMR), given by $TMR = (R_{AP} - R_P)/R_P$. There are two

types of MTJ devices, one where magnetization of the ferromagnetic layers lies in the plane of the layer is called in-plane MTJ (IPMTJ) and other where the magnetization is perpendicular to the plane of the layer, is called perpendicular MTJ (PMTJ). So far, most the devices that have been fabricated are with the IPMTJ and most of the models in the literature are developed for this type of device. It has been observed that PMTJ devices show a higher TMR than IPMTJ, and recently they are being developed more widely, though very few models have been developed so far for this type of MTJ device [151, 152].

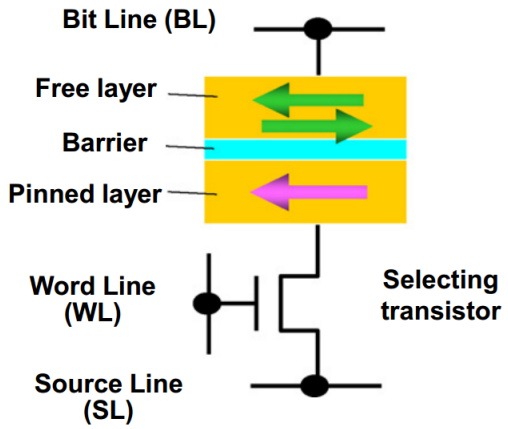

Figure 4.10 Basic structure of a memory cell with an in-plane STT-MTJ device. The alignment of magnetization in the free layer which is controlled by applying appropriate programming currents decides the overall resistance (reproduced from Kawahara, *et al.*, 2012).
Source: [151].

The basic principle of operation of STT-RAM is the exchange of angular momentum between spins of local magnetization of the layer and that of free electrons, that cause the magnetic domain walls to change their direction [151, 153]. When electrons flow from a PL to FL, only the electrons with same spin as the PL pass through to the free layer and remaining ones get filtered out. In the free layer, the spin polarized current exerts spin torque on the magnetization of the FL and causes it to switch its orientation parallel to that of the PL if the amount of current exceeds a certain threshold. Thus, the resistance of the device is lowered. For the reverse

switching case, i.e., when the charge carrier electrons travel from free layer to the PL, the electrons with opposite spin are reflected back to the FL, causing the FL magnetization to align in anti-parallel direction with respect to the PL.

The overall resistance of the MTJ device depends of the oxide barrier thickness (t_{ox}) and the interfacial effect between the oxide barrier and the ferromagnetic layers [152]. The parallel state resistance is calculated as:

$$R_P = \frac{t_{ox}}{F \times \bar{\phi}^{1/2} \times Area} \times \exp\left(1.025 \times t_{ox} \times \bar{\phi}^{1/2}\right) \tag{4.9}$$

where, $\bar{\phi} = 0.4$ is the potential barrier height of crystalline MgO, and the factor F depends on the resistance-area product ($R.A$) value of the MTJ.

The dynamics of the magnetization vector ($\mathbf{M}$) of the MTJ device is described by the LLGS (Landau-Lifshitz-Gilbert-Slonczewski) equation [153, 154]. This incorporates the torque effect due to the Zeeman energy that causes rotation of $\mathbf{M}$ around the magnetic field. Anisotropic torque is the result of interaction between individual magnetic moments and local electric field. $\vec{M}_D$ is the damping torque that causes an attenuation of the precession of the magnetization around the effective field ($\vec{H}_{eff}$). The term $\vec{M}_S$ represents the spin torque, calculated using the current flowing through the device, with the efficiency η depending of the current direction [155]. The complete LLGS equation is given as [154]:

$$\frac{d\vec{M}}{dt} = \frac{dM_Z}{dt} + \frac{d\vec{M}_A}{dt} + \frac{d\vec{M}_D}{dt} + \frac{d\vec{M}_S}{dt} \tag{4.10}$$

$$\frac{d\vec{M}}{dt} = -\gamma\mu_0\vec{M} \times \vec{H}_{eff} - \gamma\frac{2K}{M_s^2}(\vec{M}.\vec{\mu}_{ea}).(\vec{M} \times \vec{\mu}_{ea}) + \frac{\alpha}{M_s}\vec{M} \times \frac{d\vec{M}}{dt} + \eta\frac{\mu_B I}{eV} \tag{4.11}$$

Here, γ is the gyromagnetic factor, K is the anisotropic constant, $\vec{\mu_{ea}}$ is the unit vector along the easy axis, α is the damping factor, μ_B is the Bohr magneton, M_s is the saturation magnetization and V is the volume of the free layer. The effective magnetic field $\vec{H}_{eff}$ is the sum of the intrinsic damping field $\vec{H}$ and the thermal

fluctuation $\vec{H}_f$. The thermal fluctuations within the MTJ cause stochastic switching of the device state when presented with sub-critical programming input.

4.2.2 Previous Compact Models for STT RAM Devices

Different research groups have published models for the STT-RAM MTJ device, capturing various aspects of the device physics [152, 155–161]. There are two approaches followed by different groups to develop a compact model for STT-RAM device. The first one involves modeling the device switching behaviorally based on the magnitude of the programming current. The set of equations in this model define the time needed for the device to switch its state depending on the region of the current flowing through the device. Critical current needed to cause switching at absolute zero temperature is [156]:

$$I_{c0} = \frac{2e\alpha M_s V.\left(H_K \pm H_{ext} \pm 2\pi M_S\right)}{h\eta} \tag{4.12}$$

where, α is the damping factor, M_s is the saturation magnetization, H_K is the anisotropic field, H_{ext} is the external field, h is the Planck's constant and η is the spin efficiency factor. When $I < I_{c0}$, switching that occurs is stochastic in nature and the critical current needed at a pulse width of t_p for switching as given by the Neel-Brown model is [158, 159]:

$$I_c = I_{c0}.\left(1 - \frac{1}{\Delta}\ln\left(\frac{t_p}{\tau_0}\right)\right) \tag{4.13}$$

where, τ_0 is the minimum time needed for magnetic reversal, $\Delta = E_b/k_BT$ is the thermal stability factor. The switching probability is then given as [158]:

$$P_{sw}(t_p, I) = 1 - \exp\left(\left(\frac{-t_p}{\tau_0}\right)\exp\left(-\Delta\left(1 - \frac{I}{I_{c0}}\right)\right)\right) \tag{4.14}$$

When the applied current exceeds the critical value, the precession of the magnetization vector dominates over the thermal fluctuations and causes switching.

Sun's model defines the switching time for a given current $I > I_{c0}$ as [158]:

$$\tau_2 = \frac{1}{\alpha\mu_0\gamma M_S}.\frac{I_{c0}}{I - I_{c0}}.\ln\left(\frac{\pi}{2\theta_0}\right) \tag{4.15}$$

Some groups have also followed this behavioral modeling approach and incorporated thermal fluctuations as variations in the initial temperature T [152]. They use this to model the switching stochasticity of the device.

The second approach that most groups have followed is modeling the dynamics directly using the LLGS equation (Equation 4.11) [154, 155, 157, 160–162]. There have been several demonstrations of building a model in SPICE or Verilog-A that solves the stochastic LLGS (sLLGS) equation was described by [157, 160]. Their model consists of the LLG solver and a thermal variation block that evaluate all the three components of the magnetization vector (M_x, M_y, M_z). On introduction of thermal noise H_f, the LLGS equation Equation 4.11 becomes a stochastic differential equation (SDE). Methods such as implicit mid-point, Heun and Runge-Kutta4 (RK4) have been used as methods to solve the sLLGS equation [163, 164]. They also report the limitation of SPICE based solvers in accurately solving the sLLGS equation.

Papers surveying the different MTJ modeling approaches report the different merits and drawbacks of these methods [160,165,166]. In terms of speed, it is reported that the behavioral modeling approach is faster while other survey reports state LLGS based approach is faster when using the Cadence simulator [160, 166]. However, [160] reports that LLGS based dynamic models accurately represent the behavior of the device and can be used to incorporate thermal effects for demonstrating stochastic switching.

The model developed by [155], uses just one equation to describe the complete dynamics of the magnetization, including threshold switching in the device. Their model is developed for an in-plane STT MTJ device and there is no temperature evaluation block in the model. The 3-dimensional LLGS equation (Equation 4.11)

is simplified to one dimension after converting the Cartesian representation into spherical coordinate representation. The spin torque and damping torque act in the plane along the easy axis of the free layer, with an angle θ between the two. Anisotropic ($\vec{M}_A$) and Zeeman ($\vec{M}_Z$) torque act in the plane perpendicular to the easy axis, with an angle ϕ in between them. Thus, Equation 4.11 can be separated into two planes as follows:

$$M_s \frac{d\phi}{dt} = -\gamma \left(\mu_0 M_s H \sin\theta + 2K \sin\theta \cos\theta\right) \tag{4.16}$$

$$M_s \frac{d\theta}{dt} = \alpha M_s \frac{d\phi}{dt} + \eta \frac{\mu_B I}{eV} \tag{4.17}$$

$$M_s \frac{d\theta}{dt} = -\alpha\gamma \left(\mu_0 M_s H \sin\theta + 2K \sin\theta \cos\theta\right) + \eta \frac{\mu_B I}{eV} \tag{4.18}$$

Here, Equation 4.18 is the result of substituting Equation 4.16 in Equation 4.17. This SPICE model, which uses Verilog-A components is computationally simple. Hence, we chose this model as the starting point of our work.

4.2.3 Compact Model for STT-RAM Device

The compact SPICE model developed by [155] employ dependent sources and a capacitance to model the transient behavior of the magnetization vector based on the LLGS equation (Equations 4.11 and 4.18). The model is assumed to have the pinned layer at the bottom electrode, connecting the drain of the access device and free layer as the top electrode. Thus, a positive current flowing from the bit line (BL) to source line (SL) would cause the device to enter the parallel state with a low resistance of R_P, while the other direction would cause switching to the anti-parallel state (R_{AP}). As reported in [155], their model is validated with the experimental device data from [167] for the critical current amplitude versus input pulse width. They also validate the change in device resistance values as a function of programming current with device data from [168,169]. The model results agree with the experimental values for various oxide thickness and radii values of the MTJ device. We used the compact

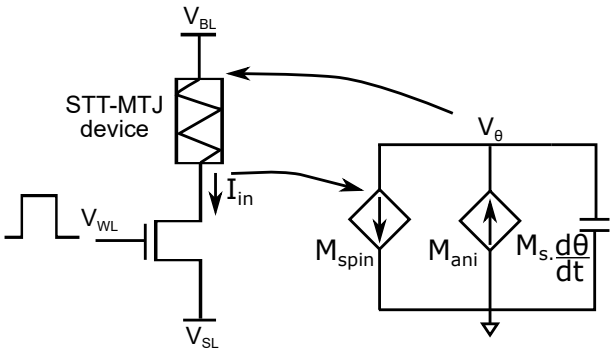

Figure 4.11 Compact model of the STT-RAM device, with bit cell connected in a 1T-1MTJ configuration. The magnetization angle θ of the device is calculated by the auxiliary circuit and is used to evaluate the device's final resistance R. This model is adapted from the one described in Xu, *et al.*, 2014.
Source: [170].

model of the MTJ STT-RAM device designed in Verilog-A as discussed in [170]. The model is described by reducing the LLGS equation to its one-dimensional equivalent form. The model in Figure 4.11 implements the different terms of the one-dimensional LLGS equation (Equations 4.19 and 4.20) as dependent current sources in different branches of an auxiliary circuit in the model. The dependent source M_{spin} computes the contribution of the torque due to spin-polarized current I_{MTJ} as:

$$M_{spin} = \eta \frac{\mu_B I_{MTJ}}{eV}, \tag{4.19}$$

where η is the efficiency factor whose value depends on the direction of current and V is the volume of the free layer [153]. The second current source M_{ani} evaluates the term corresponding to the Anisotropic, Damping, and Zeeman torques as:

$$M_{ani} = -\alpha\gamma \left(\mu_0 M_S H \sin\theta + 2K \sin\theta \cos\theta\right), \tag{4.20}$$

where α is the damping constant, γ is the gyromagnetic ratio, K is the anisotropic constant, H is the intrinsic magnetic field, M_S is the saturation magnetization and θ is the magnetization angle. The model parameters tuned to match the experimentally observed characteristics are listed in the appendix Table B.2. The base case value of

the device resistance is calculated using the magnetization angle θ as:

$$R = 2R_P \frac{(1+TMR)}{(2+TMR+TMR.\cos(\theta))} \tag{4.21}$$

where, R_P is the resistance of the device in the parallel state (Equation 4.9). The TMR value is kept at 1 for all the simulations reported here.

We have modified the model to make it well-posed by making use of smooth step functions (F_w), which prevent abrupt bias dependent switching [122, 123].

$$F_w(x) = 0.5\left(x/(\sqrt{x^2+s})+1\right) \tag{4.22}$$

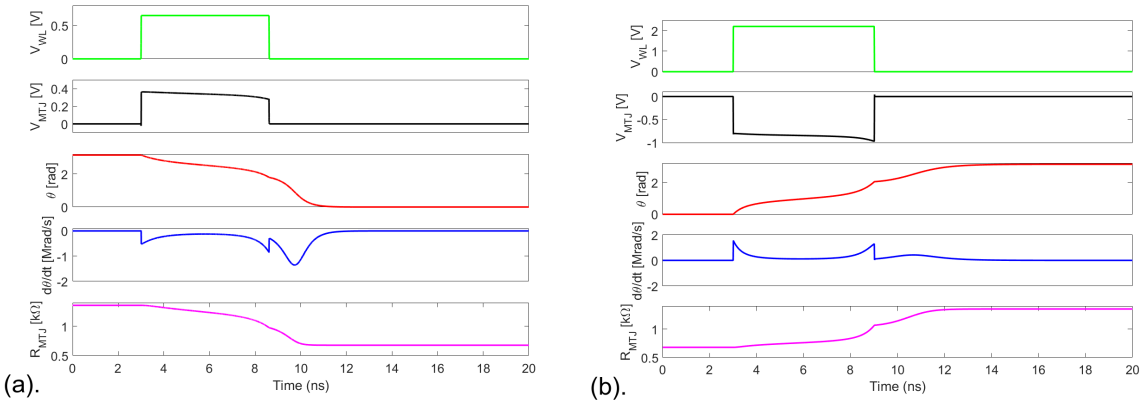

Figure 4.12 (a). Simulation waveforms when the input is above the critical point, with $V_{WL} = 0.70\,\text{V}$ and $t_{pw} = 6\,\text{ns}$ causing a deterministic switching from a high (R_{AP}) to low resistance (R_P) state. The signal $d\theta/dt$ is always less than zero. (b). Simulation waveforms for input above the critical point, with $V_{WL} = 2.2\,\text{V}$ and $t_{pw} = 6\,\text{ns}$ causing a deterministic switching from a low (R_P) to high resistance (R_{AP}) state. The signal $d\theta/dt$ is always greater than zero.

4.2.4 Simulation Results

Figures 4.12 (a) and (b) show the basic simulation waveforms for the two directions of switching. The different panels in these figures show the waveforms of the applied programming voltage (V_{WL}), the voltage drop across the device (V_{MTJ}), θ, derivative of θ ($d\theta/dt$) and the resistance (R_{MTJ}) as a function of θ when the applied voltage and

pulse width is greater than the critical value in each of the two switching cases. The model is tuned for an MTJ device with a radius of 65 nm and an oxide thickness of 1.2 nm. The parallel and anti-parallel state resistances are 677 Ω (calculated using Equation 4.9) and 1.35 kΩ (using Equation 4.21 for $\theta = \pi$), respectively. Figure 4.13(a) shows the steady state behavior of our well-posed compact model, with two stable states (HRS and LRS) when simulated in the DC mode in HSPICE.

4.2.5 Stochastic Switching

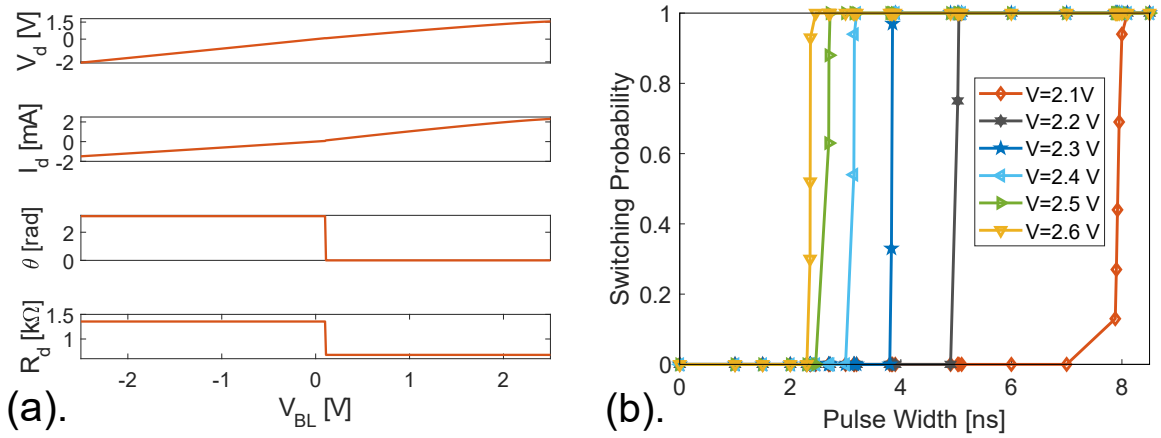

Figure 4.13 (a). Steady state response of the STT-RAM compact model obtained by varying the bitline voltage. Under this condition, the state variable θ and the device resistance remain at either high or low states when the applied voltage is negative or positive, respectively. (b). Probability of stochastic switching in the compact model, which varies with the applied input and pulse width.

The STT-RAM device shows inherent thermal fluctuations, which cause it to switch its state stochastically for sub-critical inputs [171]. This particular feature has been employed to model stochastic synapses in neuromorphic architectures [159, 172]. We have modeled this feature as an additive noise to the term M_{ani} in the expression 4.20 as, $H_{noise} = (rnd\%2) \times \sin\theta \times K_n$. Figure 4.13(b) shows the stochastic switching behavior of our compact model when sub-critical programming input is applied. Based on the amplitude of the applied voltage and its pulse width, we can generate a random set of bits in an array of 'N' such STT-RAM devices.

4.2.6 Modeling Reliability in STT-RAM Devices

Similar to PCM model, we have also incorporated stuck-at faults and resistance variability in the STT-RAM model. While STT-RAM device shows higher endurance ($\sim 10^{12}$ cycles) than the other two NVM devices, PCM and RRAM, reliability issues affect this device due to process induced parameter variations or thermal noise [115].

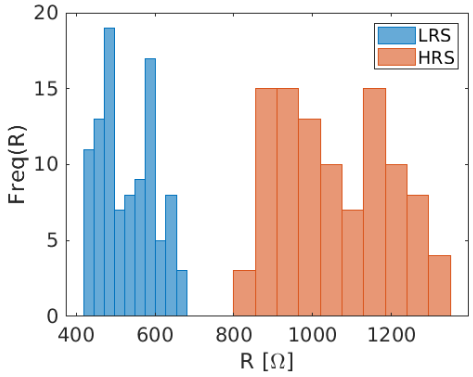

Figure 4.14 Resistance distribution for the two states of the STT-RAM model, mean high resistance at 1.35 kΩ and mean low resistance at 677 Ω.

We model the resistance variation by introducing a scaling factor, which is randomly set, in the resistance calculation expression of the model. The device resistance in a deterministic case is given by [155]:

$$R = 2 \times Rr \times (TMR0 + 1)/(TMR0 + 2 + TMR0 \times \cos\theta) \tag{4.23}$$

For the purpose of demonstrating resistance variability, we can modify the resistance expression as:

$$R = 2 \times (Rr/\$rnd) \times (TMR0 + 1)/(TMR0 + 2 + TMR0 \times \cos\theta) \tag{4.24}$$

The model shows an On-OFF ratio of 2, with nominal value of low resistance state (LRS) being 677 Ω and that of high resistance state (HRS) being 1.35 kΩ. Figure 4.14 shows the resistance distributions for the two states.

An STT-RAM memory cell also exhibits stuck-at faults when the resistive bridge short circuits or the connection between the MTJ and the access transistor is tied to either VDD (stuck-at 1) or ground (stuck-at 0) [115]. In our model, we introduce these hard faults as parameters to the spin torque term of the 1D LLGS equation. The spin-torque current is given as:

$$M_{spin} = F_{w1} \times F_{vw1} \times 1.94 \times 10^{18} \left(\frac{r_0}{r}\right)^2 \times I_{in} + F_{w2} \times F_{vw2} \times 1.29 \times 10^{18} \left(\frac{r_0}{r}\right)^2 \times I_{in} \quad (4.25)$$

To incorporate stuck-at faults in the STT-RAM model, the flag variables *StuckReset*

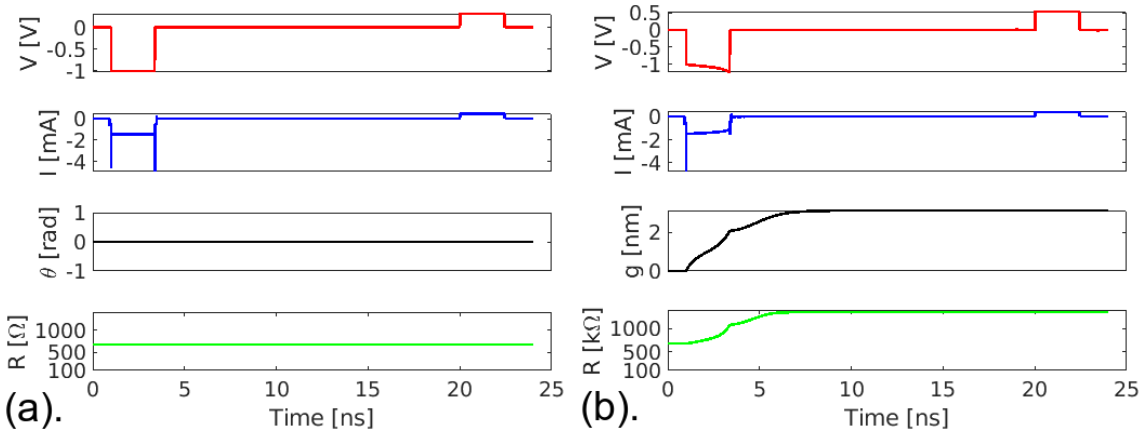

Figure 4.15 (a). Stuck-at 0 (low) faults simulation in the STT-RAM model. The first pulse is applied to change the initial state to HRS, and the subsequent pulse programs the device in LRS. It can be seen that the model remains at LRS throughout the entire simulation duration. (b). Stuck-at 1 (high) fault simulation in the STT-RAM model. The first pulse is applied to change the initial state to HRS, and the subsequent pulse programs the device in LRS. It can be seen that the model remains at HRS even after the second pulse is applied.

and *StuckSet* are introduced in the above equation as follows:

$$\begin{aligned} M_{spin} = {} & StuckReset \times F_{w1} \times F_{vw1} \times 1.94 \times 10^{18} \left(\frac{r_0}{r}\right)^2 \times I_{in} \\ & + StuckSet \times F_{w2} \times F_{vw2} \times 1.29 \times 10^{18} \left(\frac{r_0}{r}\right)^2 \times I_{in} \end{aligned} \quad (4.26)$$

We model the stuck-at faults as being uniformly distributed in an array of N such devices, hence the flags *StuckSet* and *StuckReset* are set to 0 randomly based on

the probability passed from the top level netlist. If an instance of the model exhibits either of the two stuck-at faults, the corresponding term in the equation 4.26 is turned off, hence preventing the model from transitioning to the opposite state. Figures 4.15 (a) and (b) show the simulation waveforms when the model exhibits stuck-at 0 and stuck-at 1 fault, respectively.

4.3 Resistive RAM

There have been multiple demonstrations of large storage memory arrays using RRAM devices either as planar array [173, 174] or multi-layer 3D array that have achieved high density and improved energy efficiency for computing [175, 176]. Moreover, the RRAM devices have also been demonstrated mimicking the biological mechanisms of precise timing instant based synaptic weight updates, in a network of biologically plausible neurons, also called the spiking neural networks (SNNs) [177, 178]. It was demonstrated that compared to the PCM based arrays, the RRAM based array used in implementing an SNN, with $16,000$ synapses achieved close to $1000\times$ reduction in energy consumption [177].

The resistive device used in RRAM arrays is a two terminal device which consists of an oxide material sandwiched between two metal electrodes, called as the metal-insulator-metal (MIM) structure [179]. The resistance of the structure is determined based on the formation or dissolution of a conductive filament (CF) through the oxide region between the two metallic electrodes [113] and this determines the state of the memory device.

4.3.1 Previous Compact Models for RRAM Devices

Different research groups have developed models for the RRAM device, capturing various characteristics of the device [122, 180–184]. Most of these models use two standard equations to describe the physics of the RRAM device, one for current-

voltage relation and second one for describing the gap growth dynamics. The models reported by many research groups use these two set of equations to describe the device [180–182] . The models in [181,183] also incorporate the stochastic switching behavior. The one presented in [184] evaluates the temperature by solving the heat equation in the conductive filament (CF). While these models capture the basic physics well and also perform well in transient analysis, they however, suffer from discontinuities and issues in their DC analysis [122].

In order for the model to be compliant to the requirements of NEEDS group and represent the dynamics as a system of continuous Differential Algebraic Equations (DAE) [123], we started by using the Stanford model [182]. This model incorporated the basic device current and gap dynamics along with a simple linear expression for temperature rise in the model. The model was further modified by the group at Berkley to make it compliant to the DAE format [122]. However, the Berkeley model does not incorporate temperature, and is assumed to be constant at the room temperature (298 K). So, we tuned this model to incorporate the temperature calculation and also adjusted its internal parameters to meet the current levels of an actual demonstrated device. To fit our compact model, we used the device data reported in [185]. This fabricated device was demonstrated in a large array having a size of 16 Mb [173] and was further used in implementing a binary neural network [186].

4.3.2 Model Design

Our compact Verilog-A model of the RRAM device is based on the model presented in [182]. The basic physics of the device is captured by the following series of differential and algebraic equations. When a voltage V is applied across the device, the current flowing through the oxide layer, I is the tunneling current [180], which

depends exponentially on the tunneling gap g as:

$$I = I_0 \exp\left(-\frac{g}{g_0}\right) \sinh\left(\frac{V}{V_0}\right) \tag{4.27}$$

Here, g_0 and V_0 are the fitting parameters. The tunneling gap, g which is the distance between the tip of the CF and opposite electrode, is the internal state variable in the device determining its resistance [182]. The dynamics of this variable is given by the following differential equation:

$$\frac{dg}{dt} = \nu_0 . \exp\left(-\frac{E_a}{kT}\right) \sinh\left(\frac{\gamma . a_0 q V}{t_{ox} k_B T}\right) \tag{4.28}$$

where, γ is the local field enhancement factor dependent on g, the tunneling gap variable, as:

$$\gamma = \gamma_0 - \beta g^3 \tag{4.29}$$

The terms a_0, E_a and t_{ox} in Equation 4.28 are the atomic spacing, activation energy for vacancy generation and oxide thickness, respectively. β is the switching fitting parameter as mentioned in [182]. The tunneling gap variable g is restricted to vary only between the minimum (g_{min}) and maximum (g_{max}) values within the oxide thickness. The temperature rise within the device is evaluated as [182]:

$$T = T_0 + V \times I \times R_{th} \tag{4.30}$$

where R_{th} is the thermal resistance of the device and T_0 is the room temperature. The velocity of the gap variable, g exponentially depends on the temperature, as shown in Equation 4.28.

We have modeled the RRAM device in Verilog-A and have modified it to conform to the guidelines presented in [122, 123, 187]. These guidelines state that the device physics should be represented by a set of continuous DAEs and be mathematically well-posed. Such a model is defined for operation at all theoretically

possible voltage values. Additionally, the model must also give a valid output for a constant DC input (i.e., must be valid during DC analysis) as the DC solutions are the starting point for all kinds of analyses. For the purpose of aiding convergence for the circuit simulators, the guidelines in [122] also recommends the use of G_{MIN} in the current expression. This ensures that the two terminals of the device are always connected with a finite resistance, to prevent any singularity during circuit simulation. The current voltage relation is modified as:

$$I = I_0 \exp\left(-\frac{g}{g_0}\right) \sinh\left(\frac{V}{V_0}\right) + G_{MIN}.V \tag{4.31}$$

This methodology also ensures that the internal variables of the model, such as tunneling gap, is always bounded within its physical boundaries of the CF length. The gap dynamics as in Equation 4.28 represents the state of the device at each point in time. The following subsection defines the functions that are used along with the gap dynamics in order to make it smooth even at the boundaries (i.e., when g reaches g_{min} or g_{max}). Figure 4.16 (a) shows the schematic of the model and Figure 4.16 (b) shows the 1T1R configuration of using the model in a circuit used in our simulations.

4.3.3 Clipping Functions

We present the gap dynamics as in Equation 4.28 here again for reference by denoting the derivative as f_2:

$$f_2 = \frac{dg}{dt} = \nu_0 . \exp\left(-\frac{E_a}{kT}\right) \sinh\left(\frac{\gamma . a_0 q V}{t_{ox} k_B T}\right) \tag{4.32}$$

To account for the boundary limits on g, ($g \in [g_{min}, g_{max}]$), the function f_2 is modified as:

$$f_2^*(V, g) = f_2(V, g) + F_1(V, g) + F_2(V, g) \tag{4.33}$$

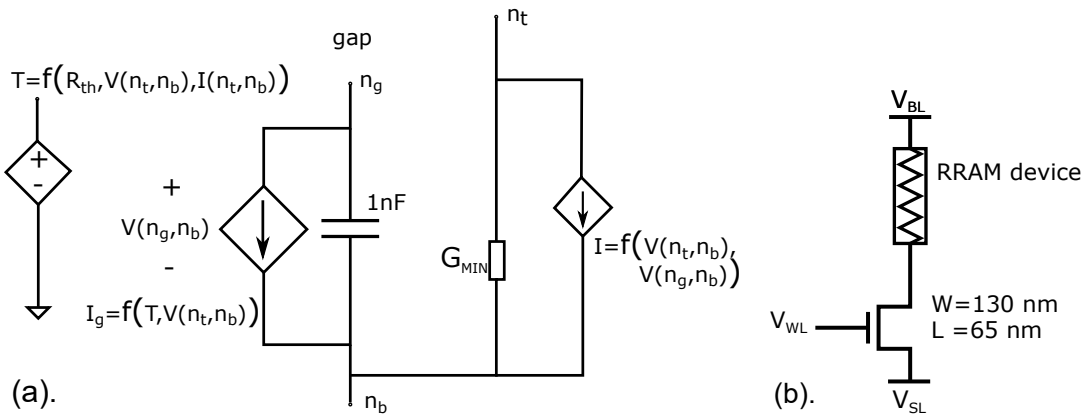

Figure 4.16 (a). Detailed schematic of the Verilog-A model of a RRAM cell. The model circuit is based on the one described in Wang, *et al.*, 2016. The nodes n_t and n_b represent the top and bottom terminals of the RRAM device. The node n_g represents the internal gap variable. (b). High level schematic of circuit used for simulating the PCM model. We use the 65 nm NMOS transistor from the PTM library to simulate the access device.
Source: [122].

where F_1 and F_2 are clipping functions given as:

$$F_1(V,g) = \left(\exp\left(K_{clip}.(g_{min}-g)\right) - f_2(V,g)\right).F_{w,1}(g) \tag{4.34}$$

$$F_2(V,g) = \left(-\exp\left(K_{clip}.(g-g_{max})\right) - f_2(V,g)\right).F_{w,2}(g) \tag{4.35}$$

and the smooth step function F_w is given as:

$$F_w(x) = 0.5\left(\frac{x}{\sqrt{x^2+s}} + 1\right) \tag{4.36}$$

where, s sets the level of smoothness. The smooth step functions are called with the following parameters:

$$F_{w,1}(g) = F_w(g_{min}-g, s) \tag{4.37}$$

$$F_{w,2}(g) = F_w(g-g_{max}, s) \tag{4.38}$$

The desired steady state response of the variable g is (i.e., $f_2 = 0$) shown in Figure 4.17. The Equation 4.33 incorporates the clipping functions and provides us the desired DC response for g.

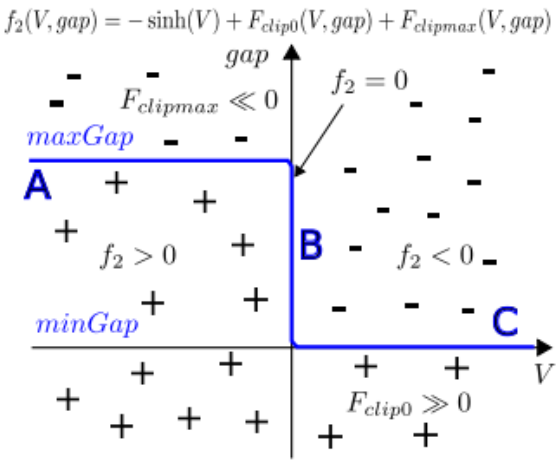

Figure 4.17 Desired steady state (DC analysis) response of the gap variable of the model, as a function of the applied voltage across the RRAM device. (Reproduced from Wang, *et al.*, 2016.)
Source: [122].

4.3.4 Simulation Results

We simulated our model in the 1T1R configuration as shown in Figure 4.16(b), which is based on the model presented in [182] and [122]. The model parameters are the same as used in the Stanford model [182]. The access device is an NMOS transistor from the Predictive Technology Model (PTM) 65 nm library. The width and length of the transistor were set at 130 nm and 65 nm, respectively. The simulation is carried out with bottom terminal of the model connected to the drain of the access NMOS transistor. To program the model to the SET state, the word line (V_{WL}) is held at 2.5 V and bitline, V_{BL} is supplied with a certain amplitude programming pulse while the source line is kept at 0 V. For RESET programming, the voltages are reversed, with bit line being grounded and source (V_{SL}) line supplied with the required amplitude programming pulse.

The fitting parameters of our model are modified to match the specifications of the $TaO_x - HfO_2$ device [185]. The model is tuned to meet the current levels of this device, which is in the order of few tens of micro-Amperes (see Figures 4.18 (a) and (b) for the I-V curve for the experimental device and our model, respectively).

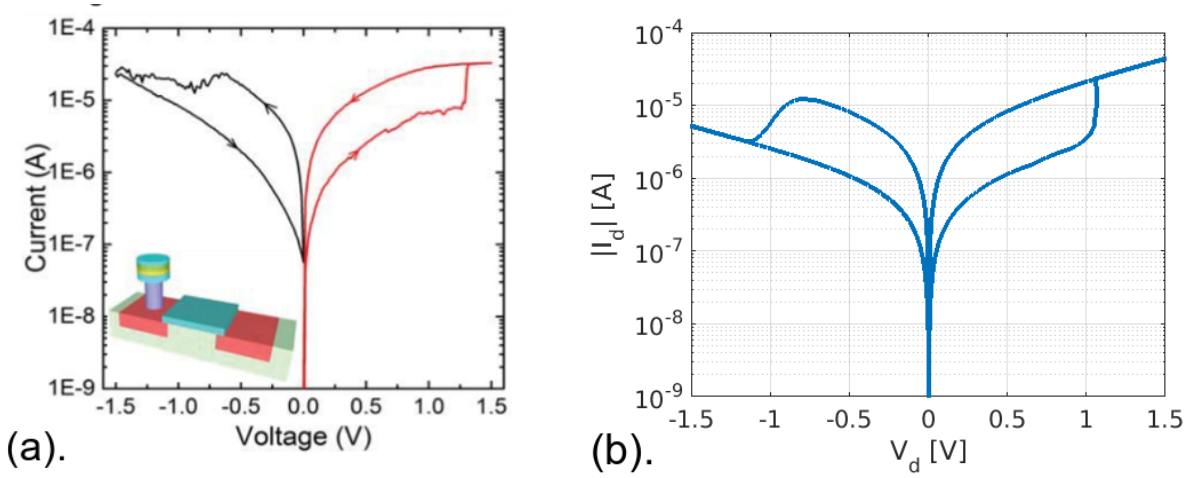

Figure 4.18 (a). I-V characteristics of an experimental TaO_x-HfO_x RRAM device used in a large memory array, as reported in Huang, *et al.*, 2015. (b). I-V characteristics of the RRAM model. It can be seen that the ON-OFF ratio of the model is around 10, which matches that of the device data.
Source: [185].

The device has been demonstrated to have an endurance of 10^6 cycles. Typical programming requirements for the device are reported as 1.75 V/50 ns for SET and 1.9 V/50 ns for RESET. The HfO_2 layer thickness is kept at $t_{ox} = 12$ nm. We also calculate the resistance during the RESET and SET states by applying a read voltage of 0.1 V (as was done in the model by [180]). The resistance in the RESET state was about 500 $k\Omega$ and in the SET state was about 60 $k\Omega$. The minimum and maximum values for the gap variable as taken as 0.1 nm and 1.7 nm as given in the Stanford model [182], representing the low and high resistance states, respectively. Figure 4.19 (a) shows the hysteresis in g when the applied voltage is slowly varied across the device. Figure 4.19 (b) shows the steady state response of g as a function of the voltage applied across the RRAM model. As explained in [122], the DC solution is not completely flat at the minimum and maximum values of gap for negative and positive voltage respectively, due to the smooth nature of the clipping functions as described in the previous section.

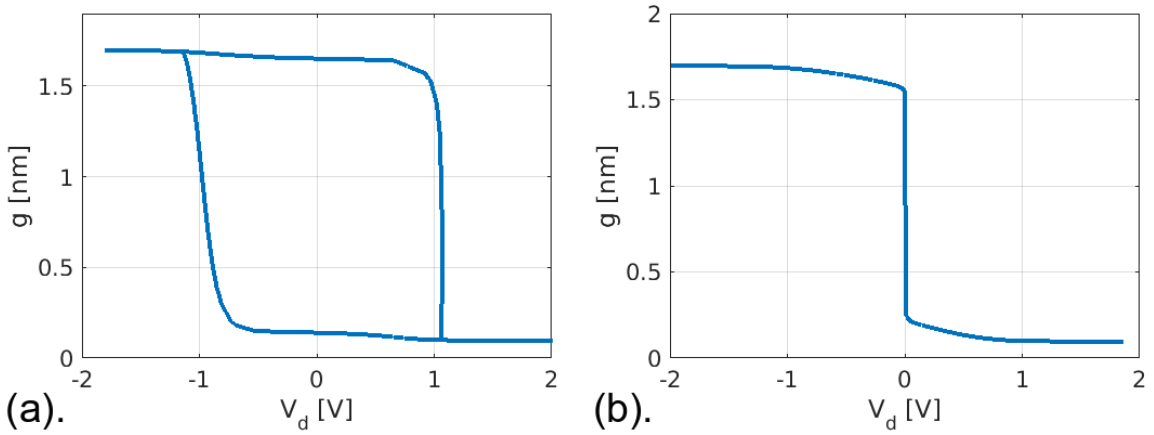

Figure 4.19 (a). Variation of gap, g, as a function of the applied voltage across the device. The gap shows hysteresis, which is the basis of memory storage capability. (b). DC analysis by varying the bitline voltage from -2V to 2V. The curve deviates slightly from the ideal behavior as in Figure 4.17 at voltage values close to zero.

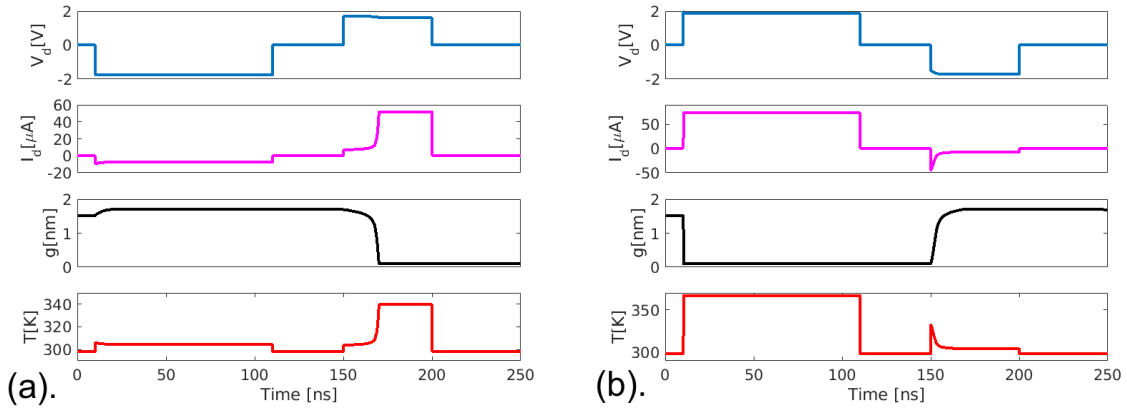

Figure 4.20 Transient simulation to programming for SET. A reset pulse is applied to the source-line V_{SL} for 100 ns and then the actual SET programming pulse of duration 50 ns is applied to the bit line V_{BL}. It can be seen that g switches from its maximum value to minimum value within about 30 ns. (b). Transient simulation to programming for RESET. A set pulse is applied to the bit-line V_{BL} for 100 ns and then the actual RESET programming pulse of duration 50 ns is applied to the source line V_{SL}. It can be seen that g switches from its minimum value to maximum value within about 10 ns.

4.3.5 Transient Analysis

The initial state of the internal gap variable is decided by the results of the DC-analysis at the beginning of transient analysis. Thus, in order to initialize the state of the device prior to programming it for a particular state (SET/RESET), we applied a pulse for the opposite state (RESET/SET, respectively). Figures 4.20 (a) and

(b) show the different waveforms for SET and RESET programming starting from a RESET and SET state, respectively. The different panels in the graphs show the voltage across the device V_d, current I_d, internal variable g and device temperature T from top to bottom. We also studied the minimum pulse width required as a

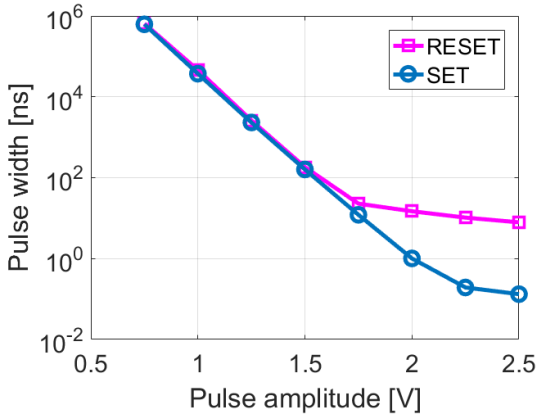

Figure 4.21 Pulse width as a function of applied programming pulse amplitude. It can be seen that there is an exponential dependence of the switching time as a function of amplitude for both SET and RESET.

function of the programming pulse amplitude as shown in Figure 4.21 and adjusted the switching speed to achieve a state transition within 50 ns for programming voltages above 1.7 V.

4.3.6 RRAM Reliability Modeling

Resistive memory (RRAMs) devices are increasingly being used in designing efficient neuromorphic hardware, and have been demonstrated to mimic the pulse timing based conductance programming, akin to spike timing based synaptic plasticity modifications in biological neurons [188–190]. However, with increased need to scale down the devices in high density arrays, different types of reliability issues occur in the NVM devices. Typically, the resistance levels of RRAM devices show a distribution of values at each level and have limited resolution [191]. Reliability is also an important consideration while realizing neural network algorithms on RRAM arrays, with multi-bit storage per device [116]. Typically faults can be either soft, i.e.,

which can be corrected with programming, or hard, wherein the device is permanently stuck at high or low resistance states. It has also been reported that hard faults, or stuck-at faults occur in at least 10% of the devices in a fabricated chip [121].

Therefore, stuck-at fault is an important design consideration, especially for devising strategies to design fault tolerant memory arrays as well as studying the hardware fault tolerance of a learning algorithm. In an RRAM device, the excess of oxygen vacancies leads to a stuck-at high fault and deficiency of these vacancies leads to stuck-at low fault [115].

We introduce resistance variability in our model, by adding small random noise terms to the bounds of the variable g as,

$$g_{max} = g_{max} + n_{max} \tag{4.39}$$

$$g_{min} = g_{min} + n_{min} \tag{4.40}$$

Here, n_{max} and n_{min} are the randomly generated noise terms added to the maximum and minimum values the gap g variable, in each model instance. g_{max} and g_{min} are the upper and lower bounds of g. A distribution of resistance values around the mean HRS and LRS values can be created when simulating multiple instances of the model.

Similar to the scheme discussed for the PCM model (subsection 4.1.4), we model the faulty state of the device as a constant value resistor between the top and bottom electrodes at LRS or HRS values, with $V(t) = I(t)/R$, where the resistance R could be R_{RESET} or R_{SET} depending on the type of fault incorporated in the particular model instance. The faults are introduced in an array of RRAM model instances stochastically based on the probability of stuck-at fault set at the top level netlist. Figures 4.22 (a) and (b) show the simulation waveforms (voltage drop, current, tunneling gap and resistance of the device), when the model has stuck-at '1' and stuck-at '0' fault, respectively.

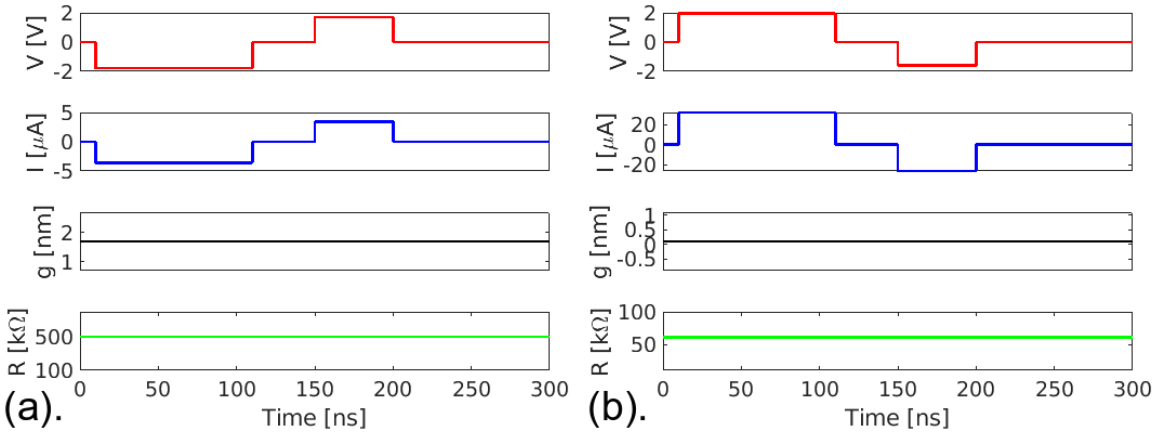

Figure 4.22 (a). RRAM model permanently stuck at its high resistance state. We apply an initial RESET programming pulse followed by a SET programming pulse. However, it can be seen that even after the second pulse, the resistance of the model remains at a high value (512 kΩ). (b). (Right) RRAM model permanently stuck at its low resistance state. We apply an initial SET programming pulse followed by a RESET programming pulse. However, it can be seen that even after the second pulse, the resistance of the model remains at a low value (61 kΩ).

4.4 Summary and Future Scope

We have developed mathematically simple and well-posed Verilog-A compact models for three emerging NVM devices – PCM, RRAM,s and STT-RAM. The models accurately capture the essential aspects of experimentally observed memory switching and reliability aspects of conductance variability and stuck-at faults. The PCM model also shows multiple resistance states depending on the programming schemes employed. The STT-RAM model also incorporates stochastic switching for sub-critical inputs.

As mentioned in the introduction, there are other reliability issues such as resistance drift, which would be an important factor in designing robust algorithms and analyzing the hardware impacts. This could be a feature to be incorporated in the presented compact models. The added reliability features could be modified as per the experimental device data to get realistic design estimates. The stochastic switching behavior of STT-RAM device could be used in realizing learning algorithms with stochastic parameters, as a random number generator. The model work presented

in this chapter could be used as a guideline in developing compact models for more emerging nanoscale devices such as ferroelectric devices, carbon nano-materials, [192], etc., to study their design prospects.

CHAPTER 5
NON VON NEUMANN COGNITIVE HARDWARE FOR SPIKING NEURAL NETWORKS

In this chapter we discuss the hardware design for SNNs. We start by presenting a CMOS digital hardware for realizing the ReSuMe algorithm [38], which was introduced in Chapter 2. We first present the results of our network optimization study and use that to develop a methodology for designing a scalable architecture for neuromorphic systems.

In the later part of this chapter, we discuss the NVM based designs for realizing inference engines and learning hardware for SNNs. We discuss memristive arrays as both analog and digital storage for synaptic weights in neural network accelerators.

We initially present our analysis on using analog memristive arrays to accelerate the inference in convolutional networks (spiking and non-spiking), which is also published in [36]. This section discusses different schemes for accelerating convolution operation in crossbar arrays and the impact of memristive non-idealities to network's accuracy. We then present the design of a non volatile memory (NVM) based neuro-synaptic core to implement a hardware accelerator for Spiking Neural Networks (SNNs). The memory array makes use of only binary conductance levels of the NVM devices. This helps in circumventing the challenges associated with these devices. Thus, our crossbar array consists of n NVM bit cells to store each of the n-bits of the network's synaptic weights. The work on implementing an inference engine with binary-storage STT-RAM arrays as discussed in the following sections has also been published in the proceedings of the ICECS-2019 [193].

5.1 Network Architecture for ReSuMe Training

We design a hardware capable of performing spike based supervised learning based on the Remote Supervised Method, that trains the synaptic weights of a spiking neuron

to generate a desired set of spike trains [59]. The original work makes use of a neural micron-circuit (NMC) network, to apply input spikes to a trainee neuron. The NMC block consists of randomly connected network of spiking neurons with certain synaptic delays, whose goal is to expand the dimension of the incoming spike train and present a rich set of spikes to the trainee spiking neuron. We present a hardware design for realizing the NMC, with the blocks needed to realize the ReSuMe algorithm on the hardware.

We first discuss the various optimizations that could be performed on the network so as to get a optimal implementation with acceptable level of training accuracy. We studied the network's training accuracy as the number of levels for representing the synaptic weights are reduced. We also looked at the dynamic range of values required by the synaptic weights, and devised the architecture based on the results of our study. The work described in this section also appears in [110].

5.1.1 Network Optimization Analyses

The ReSuMe network was analyzed for its learning speed, in terms of the number of synapses required by the output (Figure 5.1) to train for different duration of spike trains. Also, for an efficient hardware implementation, we studied the algorithm's performance at limited on-off ratios (range of weight values) and then at restricted bit representation for the weights. It was observed that just 5 bits (Figure 5.3) are sufficient to represent the weights to get a good enough learning performance and hence, a reduction in the hardware resources. Box chart plots in Figure 5.1 show that the training performance (in terms of the number of iterations needed to converge) for a particular duration of spike train, improves with increase in network size. This is because as more the number of connections from the NMC, implies more rich set of spikes at the input of the output neuron for it to train to generate the desired spike trains.

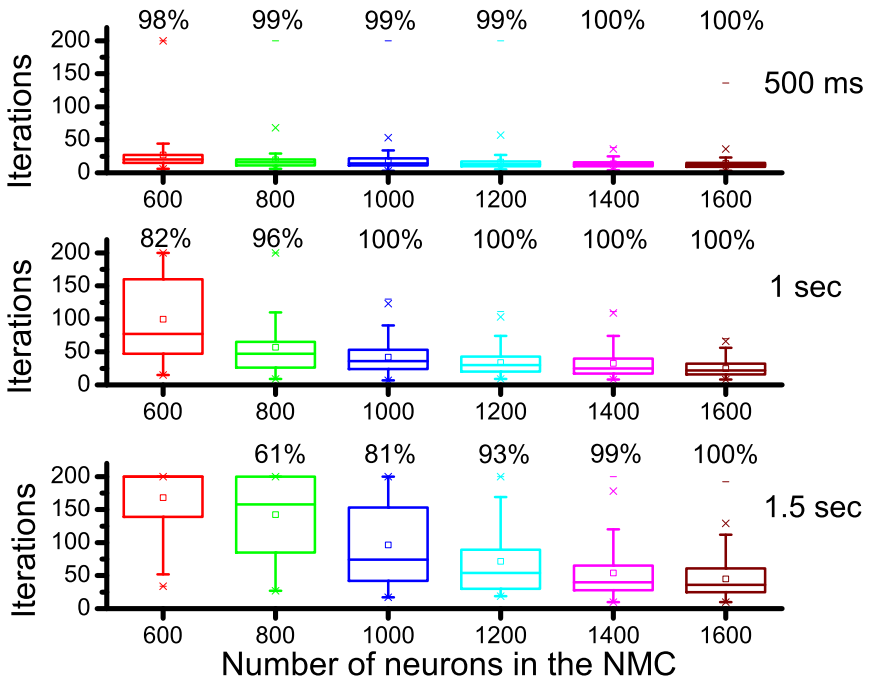

Figure 5.1 Training performance of ReSuMe at different network sizes and at varying spike train durations.
Source: [110].

We studied the effect of reduced-bit precision on the training performance. Gupta et. al, showed very less degradation on the network accuracy (with second generation ANNs) and high energy efficiency of their implementation of deep networks with just 16-bit fixed point representation [194]. We quantized the neuron dynamics during simulation to study the effect of limited precision over a full double-precision implementation. Writing the neuron differential equation (2.13) as a difference equation for every time-step n,

$$V(n+1) = V(n) + (dt/C)(-g_L(V(n) - E_L) + I_{syn}(w, n)) \tag{5.1}$$

Substituting the membrane potential, $V(n)$ as,

$$V_q(n) = \frac{V(n) - E_L}{V_T - E_L} \tag{5.2}$$

Synaptic weights are discretized as $w = w_0 \times w_q$. So, the neuron dynamics, in the discretized form can be written as,

$$V_q(n+1) = V_q(n) + Q(w_q, V_q(n)) \tag{5.3}$$

where,

$$Q(w_q, V_q(n)) = (dt/C).\left(-g_L.V_q(n) + \frac{I_{syn}(w_q, n)}{(V_T - E_L)}\right) \tag{5.4}$$

is quantized within a precision of $1/2^{20}$ following the implementation in [22]. The Figure 5.2 shows the histograms of the trained synaptic weights for 100 runs, when the network was trained with spike trains of duration 500 ms. It can be seen that the weights are log-normal distributed. It can be seen from the histograms that though the range of the trained weights in the order of 10^5, significant fraction of the weights lie only in the range 10^{-11} to 10^{-9}. The excitatory synaptic weights, have a mean of $\mu = -9.57$ and a standard deviation of $\sigma = 0.47$. The inhibitory weights have a mean of $\mu = -9.72$ and a standard deviation of $\sigma = 0.47$ on the $\log_{10}$ scale. Using this information, the training performance was studied at different on-off ratios

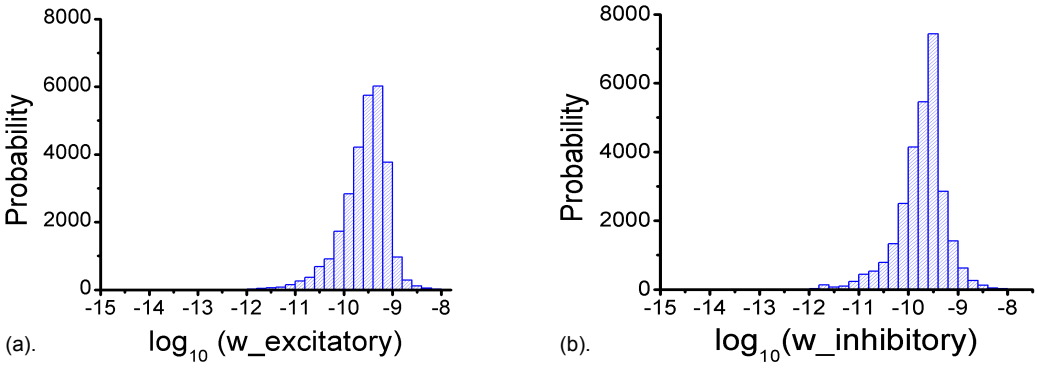

Figure 5.2 Histograms of logarithm of absolute values of the weights (a). Excitatory and (b). Inhibitory
Source: [110].

(Figure 5.3(a)), keeping the means same as the unconstrained case. The best training convergences were obtained at ratios of 100 and $10^{1.5}$. So the bit-precision experiments

were carried out for different bit lengths at these two on-off ratios (Figure 5.3(b)). At an on-off ratio of 100, and with just 5-bits of precision for the synaptic weights, it can be seen that the ReSuMe's training performance is good enough with 88% of the samples converging.

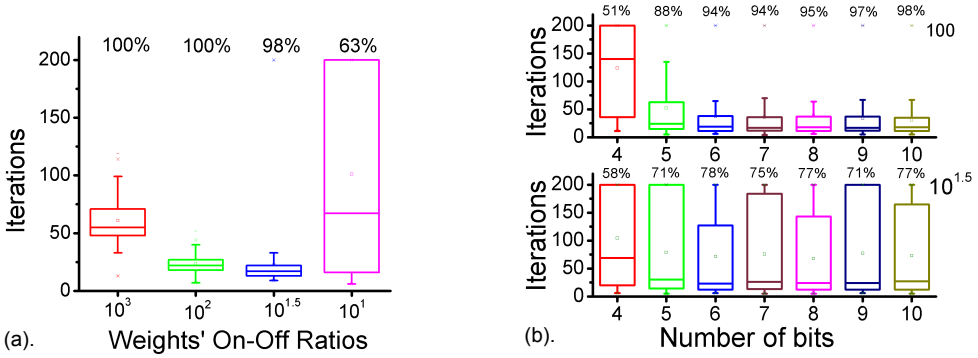

Figure 5.3 Training performance of ReSuMe at different on-off ratios and bit-precisions of the synaptic weights
Source: [110].

5.1.2 Digital Architecture

Based on the analyses in the previous section, a digital architecture for on-chip spike based learning is proposed, which is similar to the reported neuro-synaptic architectures by IBM [22, 195] and Qualcomm [196]. These chips implement programmable neuron and synaptic dynamics and use SRAM or DRAM cells to store the synaptic weights. The synaptic state can be updated in an event driven manner, depending on the instantaneous spike pattern. Most of these chips consist of large arrays of cores or small blocks that can be connected to each other through routing networks, enabling arbitrary connectivity between neurons.

Based on the results of the study in the previous section, we design a neuro-synaptic architecture for implementing the ReSuMe algorithm and its NMC network. We use the NMC network consisting of 800 neurons and each of the synaptic weights require 5 bits of storage. To make the NMC block of the ReSuMe network realizable on a crossbar architecture, we partition the block with 800 neurons into 256×256 blocks

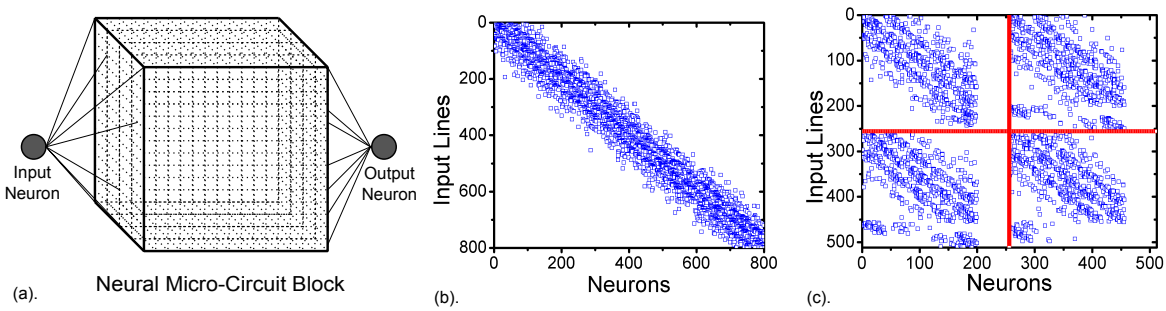

Figure 5.4 Partitioning of the NMC block on 256 × 256 sized cores.
Source: [110].

(Figure 5.4). The connectivity in the NMC is such that neighboring neurons have high probability of connection, as can be seen in Figure 5.4(b). The 3D network of the NMC, is converted to a 2D map and divided into four partitions, each consisting of 200 neurons, as seen in Figure 5.4(c). Each neuron has a fan-out (outgoing synapses) of ~ 5 . The neurons positioned at the edge of each core, have connections with the neurons in their immediate neighboring core. So, the maximum number of hops a spike would have to go through the routing network while traveling to its destination neuron in another core is just one.

The proposed design for the network of 800 NMC neurons, consists of three major components (Figure 5.5). The input block, that applies the spike streams to the network, has a mean fan-out of 240 and consists of a register file maintaining the list of the addresses (10-bit) of the fan-out NMC neurons. The output block consists of the output neuron (having a fan-in of 560), an array of registers storing the synaptic weights, and a learning module, which computes changes in the conductance values during training. The neuro-synaptic cores (inspired by [22, 195, 197]) form the third component of this architecture, housing the NMC, consist of the crossbar arrays (as seen in Figure 5.4), with 6T SRAM cells as the storage units for the synaptic weights, and the neuron circuitry. The synaptic connections within the NMC are programmed only once. However, the weights of the synapses connecting the output neuron get updated during the training. Each output synapse is associated with a 5-

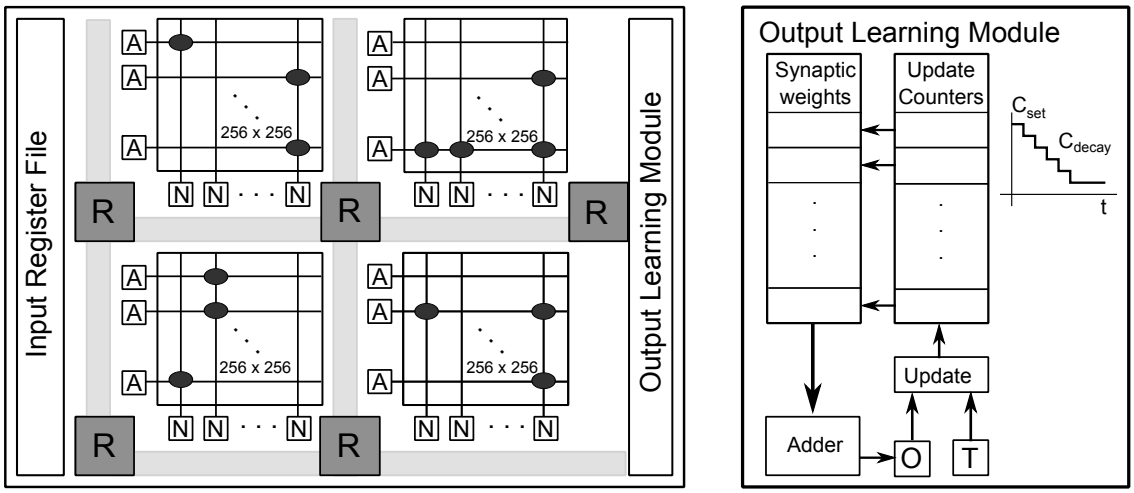

Figure 5.5 Digital architecture for ReSuMe learning. (Left) High level crossbar architecture realizing the NMC block. (Right) Learning module for calculating the weight update for the output neuron.
Source: [110].

bit down-counter (as in [198]), which is initialized to C_{set} whenever the corresponding pre-synaptic neuron spikes, and decremented by a fixed value C_{decay} at each time step to linearly approximate the exponentially decaying learning window (Figure 2.7) [199]. Whenever the output neuron spikes or there is a spike from the desired signal, the synapses which received a recent spike, get incremented or decremented, depending on the selector update block.

The basic blocks needed in the design are, the LIF block, the 5-bit STDP counter for learning and the 6-T SRAM cells. These blocks have been simulated and synthesized on 65 nm CMOS platform in order to get a first order estimates of power and area of the complete design. These numbers are then used to compute the area and power values for the entire design. For the futuristic 10 nm platform, the scaling trends reported in [198] are scaled as per the design described here.

5.1.3 Power and Area Requirements

The design was analyzed for power consumption and area required. The power consumed by the NMC cores during operation is comprised of two components – for

communicating synaptic conductance values to neurons and for updating the neuronal membrane potential, which included the neuron operation power and synaptic read and write powers ([197, 198]). The average spike rate within the NMC is close to 100 Hz. Using a time step of 0.1 ms, we get the probability of a spike issue for a neuron (p_{spike}) in any emulation time step to be ~ 0.01. For the hardware implementation, we accelerate the dynamics by a factor of 1000, so the power analysis is done by assuming a hardware clock running at $f_{avg} = 10\,\text{MHz}$, i.e., updates happen at every $0.1\,\mu$s. The power consumed during synaptic read and write operations for N neurons, consuming E joules of energy is,

$$P_{core} = E \times N \times p_{spike} \times f_{avg} \tag{5.5}$$

The power for inter core communication via the on-chip mesh network has three components: spike communication between different NMC cores, from input block to the NMC neurons and from the NMC neurons to the output neuron. It depends on the number of hops (N_{hops}) a spike packet has to travel to reach destination neurons, the number of cores (N_{cores}) a neuron communicates with and the hop distance (L_{hops}). The number of hops, B_{hops} a spike has to travel from input to the NMC cores is 2, while the number of cores, N_{cores} is 4, since the connections are uniformly distributed across the NMC block. Same is the case when a spike from the NMC has to travel to the output block. For the spikes traveling between different NMC cores, N_{hops} and N_{cores} is 1, as the cores are connected with their adjacent ones itself. The power is calculated as,

$$\begin{aligned} P_{comm} = (&10 \times N_{cores} \times N_{hops} \times L_h \times 0.2\,\text{fF}/\mu\text{m} \\ &\times V_{dd} \times V_{swing} \times p_{spike} \times f_{avg} \times N_{tot}) \end{aligned} \tag{5.6}$$

For the 65 nm analysis, V_{dd} and V_{swing} is 1 V, while for the 10 nm analysis, they are taken as 0.25 V, as reported in [198]. Tables 5.1 and 5.2 list the power consumed

and area required, respectively, by the different blocks in the design, including the four cores, at 65 nm scaled from our simulation results of a single LIF neuron (fixed-point implementation), the STDP counter and a single 6T SRAM cell. The tables also list the numbers for 10 nm, scaled as per the trends reported in [198].

Table 5.1 Power Estimates in μW for 65 nm and 10 nm

Technology node	NMC cores	Input Block	Output Block	Total
65 nm	24,500	46	724	25,270
10 nm	730	1.43	17.5	748.93

Table 5.2 Area Estimates (in μm^2) for 65 nm and 10 nm

Technology node	NMC cores	Input Block	Output Block	Total
65 nm	1,810,000	1,800	66,700	1,878,500
10 nm	111,000	200	3,800	115,000

Table 5.3 Communication Power in μW for 65 nm and 10 nm Node

Technology node	NMC to NMC	Input to NMC	NMC to Output	Total
65 nm	30	257	600	887
10 nm	0.5	4.0	9.5	14.0

5.1.4 Learning Capability of the Architecture

To quantify the learning capability of our design, the metric used is Synaptic Updates Per Second (SUPS) per Watt from the relation,

$$SUPS/W = \frac{\text{Avg no. of weight updates per synapse} \times \text{No. of synapses}}{T_d \times \text{Power consumed}} \tag{5.7}$$

The average number of synaptic updates for training $T_d = 500$ ms spike streams was ~ 20. With 560 learning synapses, we estimate that the 65 nm CMOS implementation

to be capable of 0.85 MSUPS/Watt and the corresponding projected value for the 10 nm node to scale to about 30 MSUPS/Watt.

We have shown the methodology to realize spike based learning on-chip. This study was done using the ReSuMe algorithm, which is a local learning rule. The architecture required just 0.6 mW of power and 0.115 mm^2 area to realize the 800 neuron network for ReSuMe. This study was targeted towards digital CMOS implementation. Further, we will demonstrate the realizability of such spike based learning rules on platforms with emerging nano-scale memory devices. We make use of our developed compact models for PCM, STT-RAM, and RRAM devices for implementing synapses in crossbar based architectures and evaluate the hardware metrics over a pure CMOS implementation.

5.2 Accelerating Spiking Neural Networks with Memristive Hardware

In this section, we discuss the use of a generic NVM memristive crossbar array, wherein the analog resistance levels are used to store the synaptic weights of a network. Part of the network computation which involves multiply and accumulate can be carried out within the memristor array. The work presented in this section is as published in [36].

NVM devices, where device resistance stores the state, such as phase change memories (PCMs), spin-transfer-torque (STT) RAMs and Resistive RAMs (RRAMs) can be programmed to mimic conductivity modulation of biological synapses [113, 200–203]. Numerical estimates of NVM based implementations suggest more than 25× improvement in speed-up and up to 3000× reduction in power compared to GPU based implementations [204]. NVM devices are particularly suited for processing-in-memory architectures, for instance the PipeLayer accelerator based on RRAMs for Convolution Neural Networks (CNNs) for training and testing showed significant speedup and energy efficiency over GPUs [205]. Other memristor based

crossbar implementations for CNNs showed that there is negligible loss in performance if the device has at least 4-bit resolution when used as inference engines [206]. Similarly, an extremely parallel architecture for accelerating deep CNNs has been proposed by leveraging the small size and high density of memristive devices [207]. Therefore, memristive devices offer one possible way to emulate brain's connectivity in hardware, if other non-ideal limitations of the devices can be mitigated. Good network performance can be obtained if the synaptic devices have linear and symmetric conductance response; but in reality, the device conductance is highly sensitive to programming variability and typically has finite on-off ratio with limited resolution [191]. It has been shown that conductance variability is a critical parameter that has to be optimized to maintain the ideal performance achievable in software simulations [116]. While the above mentioned efforts have studied the applicability of NVM devices with limited resolution for neural network realization, our work explores the impact of the programming variability of these devices on algorithmic performance.

We use a convolutional SNN for handwritten digit classification (which was described in Chapter 3) and describe a methodology of accelerating such a network on hardware using nanoscale memristive devices. We study the accuracy of this network as an inference engine when its synapses are implemented using memristive devices. We also benchmark the performance with respect to an equivalent second generation artificial neural network (ANN), where the trained SNN's accuracy is close to that of the trained ANN. Our results show that even at a very high device conductance variability, the accuracy for both the networks is within 1% of their respective baselines. We also study the realization of the convolution layer in our network as a complete parallel implementation by having a memristive device for each synaptic connection as opposed to having a shared synaptic array, as is done in the software and show that the parallel architecture helps in mitigating the effects of conductance variability on network's accuracy especially at higher variabilities. Thus,

this work shows the high potential for realizing efficient inference engines based on event-triggered spiking networks implemented using memristive devices.

5.2.1 Network Optimization for Hardware

We now discuss the network optimization strategies to translate the software design for energy and area efficient neuromorphic hardware platforms. Typically, such platforms have limited precision for weight storage. Previous efforts to study the impact of low-precision weights in a digital realization have shown close to 5% drop in accuracy with respect to the baseline even with 5-bits of precision for synaptic weights [109].

Memristive devices can also be used as synaptic weights in crossbar neuromorphic platforms [208]. Even though the device conductance is an analog value, the granularity to which a device can be programmed to a particular level is typically limited, resulting in a finite number of levels within the dynamic range of the device. In the following subsections we discuss our study on the impact of realizing synaptic weights with memristive devices, with an on-off ratio of 10 and a resolution of approximately 32 levels (or 5-bits). These characteristics are typical of several experimental memristive devices today.

5.2.2 Restricting ON-OFF Ratios of Synaptic Weights

To study the impact of using memristive devices as synapses in our neural network, we measure the inference accuracy by limiting the range of weight values that were obtained after training the networks in software. The learned weight values of the neural networks trained in software were limited to have an on-off ratio of 10. Figure 5.6 shows histogram of the software trained weights and the histogram after clipping them to have an on-off ratio of 10. Table 5.4 shows the accuracy and the number of non-zero connections in the feed-forward fully-connected layer of the

SNN and ANN, before and after removing the insignificant weight values. It can be seen that even after eliminating more than 50% of the trained weights from the two networks, the drop in test accuracy is negligible, indicating that sparsely connected networks are capable of delivering close to baseline accuracies.

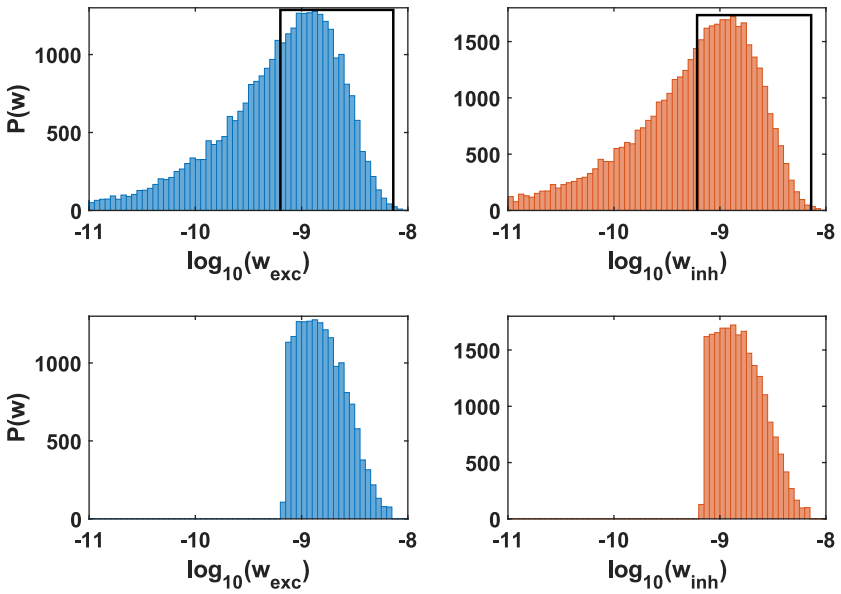

Figure 5.6 Software trained weights (upper panel) having a large range of values, are clipped such that the ratio of maximum value to minimum value (on-off ratio) of the excitatory and inhibitory weights is restricted to 10. This range of values resulted in the test accuracy to drop to 98.07% from the baseline value of 98.17% in the SNN. For the ANN, there was no drop in the test accuracy.
Source: [36].

Table 5.4 Network Accuracy during Inference with Limited On-Off Ratio of 10

Network	SNN	ANN
Non-zero synapses in baseline network	75,000	78,000
Baseline accuracy on MNIST test set	98.17%	98.10%
Non-zero synapses in the new network	30,000	27,000
Inference accuracy in the new network	98.07%	98.10%

5.2.3 Hardware Architecture

While the synapses between the fully-connected layers can be implemented using a cross-bar array in a straight-forward manner [204], the implementation of the convolution operation is the focus of our attention in this study. A convolution layer in a neural network uses a weight matrix (also called convolution kernels), whose element values are tuned to extract particular features from the input image. The software implementation of the convolution sequentially repeats the matrix across the entire input, and a dot-product of the input overlap with the weight matrix is computed. Although this computation can be accelerated on a GPU (graphical processing unit), by scheduling the computation of convolution for elements and kernels in a concurrent manner, the limited number of computation cores always limits the degree of parallelization possible, thereby limiting the overall speed.

There have been some recent efforts to implement the convolution operation using NVM based neuromorphic hardware [205–207, 209, 210]. Here, we study the impact of using the analog conductance levels in memristors, with two devices in a differential configuration per synaptic weight to realize the excitatory and inhibitory weight connections [206, 208].

5.2.4 Synapses Using Memristive Devices

In order to represent positive and negative weights (w), we use two memristive devices with conductances G^+ and G^- per synapse [208], such that

$$w = k(G^+ - G^-) \tag{5.8}$$

Appropriate scaling factor (k) is used to translate the device conductances to the range of software trained weights. Most memristive devices also exhibit gradual conductance change in one direction; hence we assume a unidirectional programming scheme [211]. The device is assumed to have an on-off ratio of 10 and roughly 32 levels

of programming resolution for realizing the synaptic weights. In this scheme, either G^+ (G^-) will be programmed to any of the allowed 32 conductance states, depending on the sign of the software weights at every synapse. For the fully connected layer in SNN and ANN, including the inhibitory weights of SNN, when the synaptic weights are zeros, both G^+ and G^- are programmed to the minimum conductance in the linear regime.

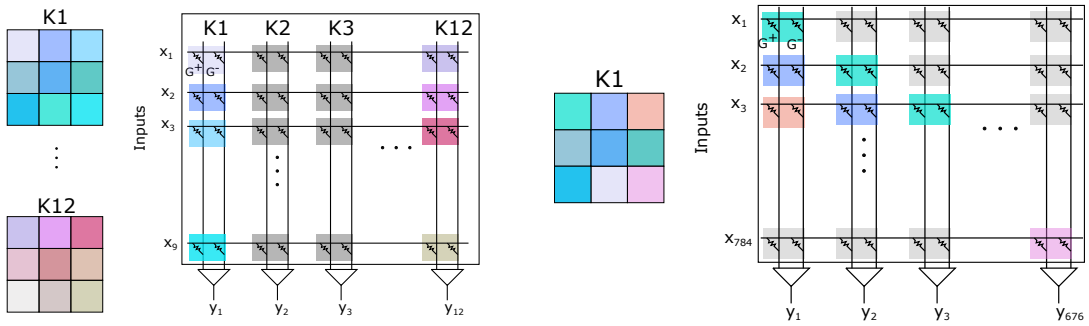

Figure 5.7 (Left) Sequential convolution in memristive crossbar array with $9 \times 12 \times 2$ devices to represent the 12 kernels used in the convolution layer, each having a 3×3 sized weight matrix. These matrices are unrolled as vectors of size (9×1). The inputs need to be presented in sections of 9 elements to obtain the output of the convolution operation. (Right) Parallel convolution in memristive crossbar array with $784 \times 676 \times 2$ devices. Each neuron in the convolution layer has 9 incoming synapses, so every column in the array has only 9 active connections. The cross-points in gray are inactive connections.
Source: [36].

5.2.5 Sequential and Parallel Convolution

We now discuss two hardware architectural schemes for implementing the convolution layer using memristive cross-bar arrays and illustrate the architectures using a convolution operation performed between an input image of size 28×28 with a 3×3 kernel, resulting in a output matrix with 26×26 elements. In the first scheme, only the 9 unique values in the convolution kernel are represented using 9×2 memristive devices [206, 209]. Figure 5.7 (left) shows the architecture of crossbar array, where the 12 kernels are laid out in 12 columns, each of length 9. Here, the inputs x_i need to be presented to the array in batches of 9 elements at a time. The outputs y_j, for

each kernel are obtained sequentially, by the application of Kirchhoff's law. A voltage signal proportional to the input quantities x_i are applied on the horizontal wires, and the resulting current through the synapses are accumulated by the sense amplifiers on the vertical wires. While this scheme uses only $9 \times 12 \times 2 = 216$ devices for the array, 676 *sequential* cycles are needed to complete the convolution across all kernels for each MNIST image.

In the second scheme, a larger memristive array is used so that all the convolution operations are completed in parallel. In this array, every connection between the input and output has a synaptic device associated with it [207]. Figure 5.7 (right) shows the architecture of the crossbar array with 784×676 synapses for a single convolution kernel output map. Such arrays need to be repeated for multiple kernel output maps. This scheme requires $2 \times 784 \times 676 = 1,059,968$ memristive synaptic devices per convolution output map. However, since each neuron in the convolution layer connects to only 9 inputs, the active number of connections in the network are only $9 \times 676 = 6084$, resulting in a sparsity of $\sim 1\%$. With this scheme, we can achieve a speedup of $676\times$ over the sequential realization. The implications of using multiple memristors, which are known to have conductance variabilities, for every synaptic connection as opposed to having a shared array is discussed in the subsection 5.2.7.

5.2.6 Programming Variability

Memristive devices are non-ideal, exhibiting significant programming variability, which may affect the network performance [116, 191]. The impact of programming variability in the network performance is studied using the parameter σ/B, where σ is the standard deviation and B is the bin-width of the conductance states obtained during programming. In order to emulate the programming variability of the devices in the simulation, a zero mean Gaussian noise of standard deviation σ is added to the programmed device conductance. We study the network's inference accuracy for

different programming variability with $\sigma/B = 0.5,\ 0.8, 1$ and 1.5. The programming variability used in this study is comparable with experimental reported values for typical memristive devices [212]. As the programming variability parameter is increased from $\sigma/B = 0.5$ to 1.5, the conductance spills over to the neighboring bins, thereby potentially affecting the network's inference accuracy.

In our simulations, this conductance variability is included in the synapses for all the layers in the parallel and sequential convolution networks. Further, we assume that the inactive devices in the convolution layers can be programmed to $0.1 \times G_{min}$ so as to minimize the effect of programming variability in the convolution kernel. Also, the programming variability associated with these inactive devices is assumed to be one-tenth of σ/B. These assumptions are based on the experimentally observed characteristics of emerging memories, whose off-state conductance can be at least 10 times lower than the typical analog range used for neuromorphic weight storage [212, 213].

5.2.7 Results

Our SNN and ANN are implemented in CUDA-C and C, respectively. The software baseline response of SNN and ANN are at 98.07% and 98.10% inference accuracy, respectively when the weights are clipped. On translating these clipped weights to the allowed conductance states of the memristive device without any variability, the inference response drops slightly to 97.99% for SNN while it remains the same for ANN. These accuracies are used as the device baseline for analyzing the response for different programming variability.

Sequential Convolution The response of sequential convolution architecture of SNN and ANN after introducing programming variability to the memristive devices is shown in Figure 5.8 (left). It can be seen that as the variability is increased, ANN suffers slightly less degradation in accuracy when compared to SNN. The input

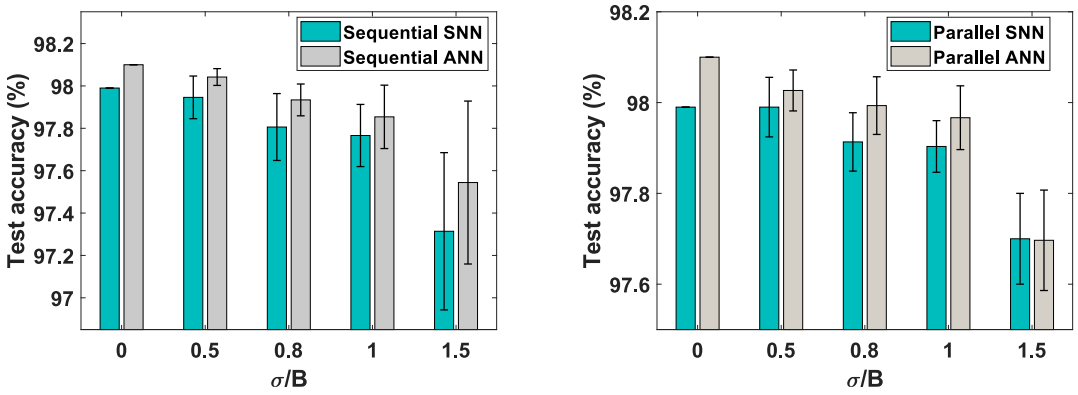

Figure 5.8 Accuracy of the spiking and non-spiking networks for sequential (left) and parallel (right) convolution as a function of the device conductance level variations, defined as the ratio σ/B, where σ is the standard deviation of the zero mean Gaussian noise and B is the bin-width of the conductance levels. In both the cases, the average classification accuracy of the SNN is close to that of the ANN within 0.1%.
Source: [36].

currents corresponding to those images that were misclassified by SNN ($\sigma/B = 1.5$) but correctly classified by ANN ($\sigma/B = 1.5$) and the device baseline networks of SNN ($\sigma/B = 0$) and ANN ($\sigma/B = 0$) is analyzed in Figure 5.9. The neuronal input

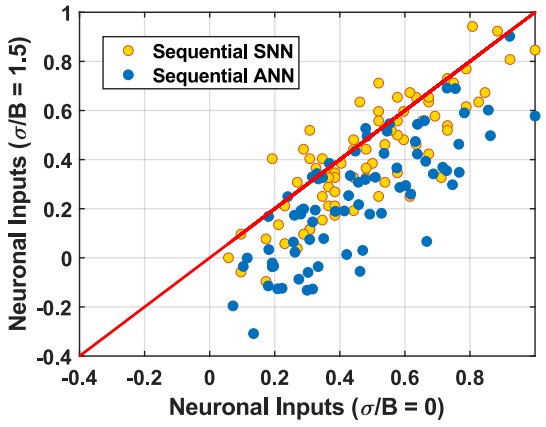

Figure 5.9 The incoming currents to the output layer neurons of both SNN and ANN for $\sigma/B = 1.5$. The x-axis corresponds to the baseline network without any programming variability ($\sigma/B = 0$), while the y-axis represents the networks with variability $\sigma/B = 1.5$. It can be seen that for SNN, input currents deviated more from the baseline when compared to ANN, resulting in the slightly higher accuracy drop.
Source: [36].

current in the SNN is obtained as an integrated value over a duration of $T = 100\,\text{ms}$,

while for the ANN it is the instantaneous DC value of the weighted inputs to each neurons. It can be seen that the deviation in the ANN from the baseline network having no variability is lesser compared to that of the SNN, which explains the lesser degradation in the accuracy of ANN (Figure 5.8).

Parallel Convolution High density packing of the memristive devices can be leveraged to implement parallel convolution giving significant speed up in hardware [207]. The convolution operation for the SNNs can also be implemented in parallel using the network architecture in Figure 5.11 (right). Similar to the sequential convolution implementation, even in the parallel network, the SNN's accuracy was close to that of the ANN within $\sim 0.1\%$ (see Figure 5.8 (right)). As can be seen from Figure 5.10, the inference accuracy of the two schemes (parallel and sequential) is comparable, although the parallel implementation shows slightly better accuracies at high device programming variability. The slightly better performance with the parallel convolution architecture may be due to the averaging and compensating effect of memristive devices in determining the output of the convolutional operation. In

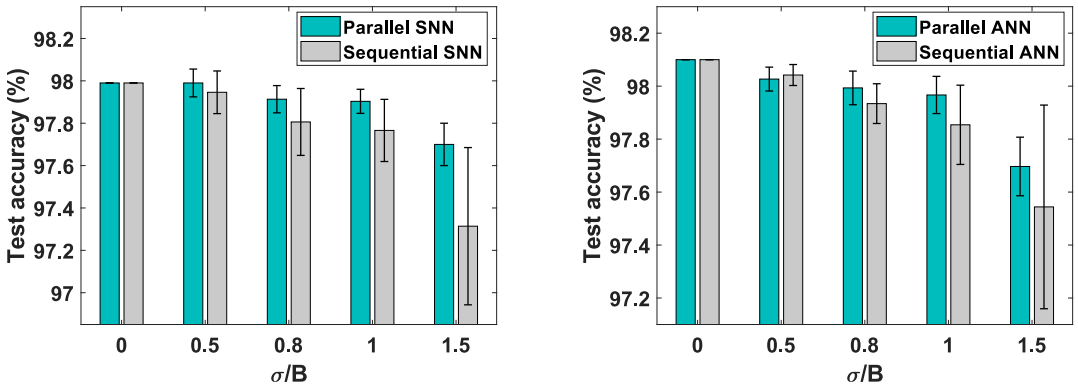

Figure 5.10 Comparison of networks' inference accuracy for sequential and parallel convolution architectures with memristive arrays in SNNs (left) and ANNs (right). *Source:* [36].

order to obtain this performance, it is important that the inactive devices in the array

are programmed to low conductance values, below the normal dynamic range used for synaptic weight representation.

5.3 Digital Cognitive Hardware Design with NVM Crossbar Arrays

As discussed in the previous section, there are several crucial challenges of variability associated with the device conductance, and the additive noise from the peripheral circuitry such as Analog to Digital converters (ADCs) and Digital to Analog Converters (DACs) that need to be taken care of while designing these architectures [26, 35, 36].

Next, we explore the scope of running inference and training in smaller bit-precisions while designing the hardware. While a 32-bit single precision computation is what is used by most computing platforms today, it has been shown that significant energy and speed benefits can be achieved in the hardware if the computation can be carried in lower bit-precisions [214]. Several works for DNNs have demonstrated running inference engines in 1 to 8-bit fixed-point formats [215–217]. There have also been several efforts in demonstrating algorithmic modifications to DNN training such as transfer learning, stochastic rounding, etc. to achieve the 32-bit floating-point accuracy while training in lower precisions [218–220]. Low-precision implementations are beneficial especially for embedded and edge devices, which run with a limited power budget and memory capacity.

Our computational neuro-synaptic core consists of a crossbar array of NVM devices, read/write peripheral circuits, and digital logic for the spiking neurons. Additionally for learning, blocks such as a multiply and accumulate unit, schedulers, are needed to perform the error back-propagation and weight updates. Inter-core communication is realized through on-chip networks by sending/receiving spike packets. The design studies are conducted using the compact models that we developed for the three NVM devices – Phase Change Memory (PCM), Resistive

RAMs (RRAM), and Spin-Transfer Torque RAM (STT-RAM), which are tuned to capture the state-of-the-art experimental results. We show that amongst all the three for realizing inference engines, the STT-RAM memory based design outperforms the other in terms of throughput per unit Watt. We further realize a large memory array with 2048 × 2048 STT-RAM devices to implement SNN and compare the same with an equivalent SRAM based design realized in [46]. Our binary storage STT-RAM based design avoids the need for expensive ADCs and DACs, resulting in a balanced array efficiency of 53%, and enabling instantiation of large NVM arrays for our core. We show that this neuro-synaptic core designed in 28 nm technology node has approximately 6× higher throughput per unit Watt and unit area than an equivalent SRAM based design. Our design also achieves ∼2× higher performance per Watt compared to other memristive neural network accelerator designs in the literature. The later part of this section also discusses schemes to extend this design to incorporate learning and compares its performance with respect to an equivalent SRAM based design.

5.3.1 NVM array for SNN Inference Engines

We design an NVM array based inference engine to realize the Binary Activation SNN (BASNN) model which was presented in Chapter 2 and was originally described in [46]. This model employs gradient descent based learning rule to adjust the network weights and is one of the simplest models to realize on the hardware. It has been demonstrated to achieve close to the state-of-the-art SNN performance on the MNIST dataset, with the best test-set accuracy being 99.4% with a convolutional network. We present the equations of this neuron model again here for reference. The discontinuous integration neuron model's spike output (a^k) located in layer k is given as:

$$a^k(t) = y_b \left(\sum_{i=1}^{N} a_i^{k-1}(t) w_{i,j}^k + b_j^k \right), \tag{5.9}$$

where y_b is the threshold function, with $y_b(x) = 1$ only if $x > \theta$ [46]. Here, θ is the membrane potential threshold. The term within the brackets represents the spiking neuron's membrane potential. The use of a straight-through estimator makes the neuronal function differentiable during training [47]. The neuron's activation derivative is:

$$a^{k\prime}(v^k(t)) = \begin{cases} \frac{1}{2\theta}, 0 \leq v^k(t) \leq 2\theta \\ 0, \ otherwise \end{cases} \tag{5.10}$$

With this weight update rule, SNN in a fully connected multi-layer perceptron (MLP) achieved the MNIST classification accuracy of 98.7% in the single precision floating point (FP32) representation. A digital design that implements the trained MLP network as an inference engine using SRAM based synaptic weights was also presented in [46]. Here, we explore the architectural design choices for implementing such networks using emerging NVM arrays.

As was demonstrated in [46], the BASNN MLP network requires only 8-bit signed fixed-point synaptic precision to achieve the floating-point baseline accuracy of 98.7% for MNIST dataset. In our scalable design, we represent each synaptic weight with signed 8-bit fixed-point precision, with upper 2 bits used for the sign and the integer part, and remaining 6 bits for the fractional part. Each WL in the array has 256 synaptic weights. The neuron block consists of an 8-bit fixed-point register to store the read-in weights and biases, and a 16-bit accumulator to store the membrane potential (v). The membrane potential is represented with signed 16-bit fixed-point precision, where the three most significant bits represent the integer part (including the sign bit). The remaining 13 bits represent the fractional part of the membrane potential. The logic consists of a comparator which uses only the upper 3 bits of the v register to compare with the threshold of $\theta = 1$. Once all the synaptic weights corresponding to the incoming spikes in a particular time-step and the biases for that

layer are accumulated, the values of v of that layer are compared with θ, to check whether to issue a spike or not. The generated spike is transmitted by the on-chip router to its destination core (Figure 5.13).

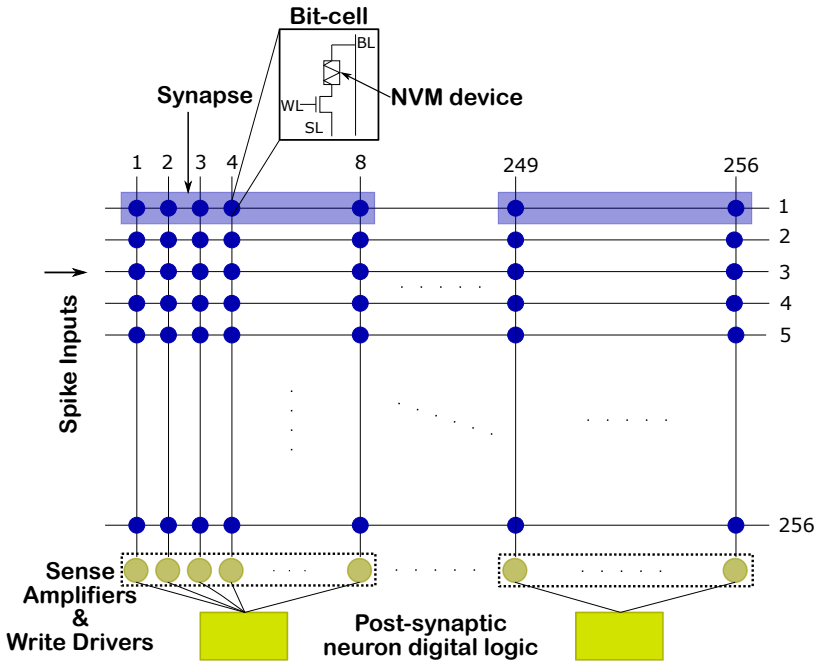

Figure 5.11 Neuro-synaptic crossbar array based hardware with 256 inputs lines, 32 output neurons and 8-bit synapses. Each output layer neuron on the post-synaptic side of the array is connected to 8 bitlines and can access the associated devices for the selected row (wordline).

For our study in evaluating the different NVM technologies, we consider a neuro-synaptic core with 256×256 NVM bit cells in 1T-1R configuration as shown in Figure 5.11. For bidirectional programming of the bipolar devices such as STT-RAMs or RRAMs device, programming waveforms can also be applied to the source lines (SLs). The input spikes, or pre-synaptic spikes, are presented to the wordlines (horizontal) of the array, and the output neurons (or post-synaptic neurons) of a particular layer are placed at the end of the bitlines (vertical) of the array. The spiking neurons are implemented as digital blocks outside the array. The read and write circuits are also placed at the post-synaptic end of the crossbar array connected to the bitlines and source lines.

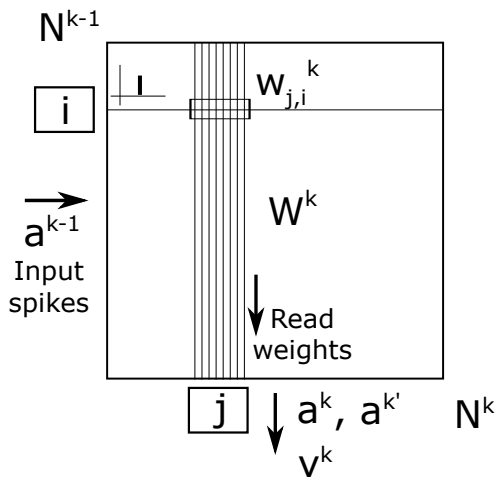

Figure 5.12 Scheme for forward propagation of input signals through the STT-RAM crossbar array. At any given time-step, a set of spikes from the layer $k-1$ arrive at the core input (WL). Each of the wordlines are processed sequentially at the respective output neurons, which read and accumulate the synaptic weights.

Figure 5.12 shows the process of propagating input spike trains (a^{k-1}) through the array for a given network's simulation time-step, where k denotes the index of a layer to be realized on the array. Each incoming spike activates its corresponding wordline, triggering a read of the synaptic weights along that row. The weights (w^k), represented in 8-bit fixed point notation are read at the post-synaptic end of the array and the neurons update their respective membrane potential values for the present time-step If a neuron's membrane potential exceeds the threshold, a spike is issued for that time-step. The duration of time-step is determined based on the number of hardware clock cycles required to read the weights corresponding to spiking inputs and update the neuron membrane potentials. The spikes issued by the post-synaptic neurons ($a^k = 1$) are then transmitted by the on-chip router to the core realizing the next layer of the network.

5.3.2 Memory Array Design

The NVM bit cell considered for this study has an area of $29F^2$ in the 1T-1R configuration [221–224]. We use a crossbar array of size 256×256 bit cells having

a memory capacity of 64 Kbit. For each of the NVM devices, our group designed the peripheral memory access circuits for reading and writing, based on the designs reported in [225–228]. The read sense amplifiers are designed as either in current mode or voltage mode. In the case of voltage mode sensing, a voltage drop across a pre-charged bitline indicates if the state is a '0' or not. In the case of current mode sensing, the current flowing through the device is compared with that flowing through a reference resistance cell. For the STT-RAM device, as the On-Off ratio is low (about 2), a current mode sensing is used. For the other two devices, which have a higher On-Off ratio of $\sim$ 10, voltage mode sensing is adopted. The read

Table 5.5 Read and Write Circuits for NVM Devices Designed in 65 nm Node

Design parameters	**PCM**	**STT-RAM**	**RRAM**
$R_{SET}\,\Omega$	$10,000$	3000	$60,000$
$R_{RESET}\,\Omega$	$1000,000$	6000	$500,000$
Read			
Area (μm^2)	21.18	41	21.18
HRS read power (μW)	2.95	1.42	2.34
LRS read power (μW)	3.07	1.51	2.67
Read latency (ns)	10	4	25
Write			
Area (μm^2)	134	324	134
HRS write power (μW)	777.2	1034	373.7
LRS write power (μW)	317.3	1027	382.98
Write latency (ns)	150	7	80

sense amplifiers and write drivers are placed along the bit lines of the array. Similar to previous crossbar array based designs [22], the horizontal wordlines connect the input (or pre-synaptic neurons) and the vertical bitlines (and SLs) connect the output

(post-synaptic) neurons of the array via the read and write circuits. The read and write parallelism of our crossbar design is 256 bits, i.e., there is one read sense amplifier and write driver circuit for every vertical bitline/source-line of the array. Table 5.5 presents the area and power numbers for the read and write circuits for the different NVM devices of PCM, STT-RAM and RRAM.

5.3.3 Design Evaluation Across NVM Devices

Table 5.6 Evaluation of Neurosynaptic Core Design Across Three Different NVMs

Parameters	**STT-RAM**	**PC-RAM**	**RRAM**
Read latency (ns)	5	10	25
Memory clock frequency (MHz)	100	50	20
Memory access power (mW)	0.53	0.92	0.79
Digital logic power (mW)	1.46	1.46	1.46
Total power (mW)	1.98	2.38	2.25
Total bit cell area (8 KB) (mm^2)	0.02	0.02	0.02
Memory peripheral area (mm^2)	0.095	0.040	0.040
Digital logic area (with 32 neurons) (mm^2)	0.02	0.02	0.02
Total core area (mm^2)	0.13	0.08	0.08
GSOPS	3.2	1.6	0.64
GSOPS/W	1610	673	285
GSOPS/mm^2	24.62	20	8
GSOPS/W/mm^2	12385	8412	3562

To evaluate the designs with these three NVM arrays, we choose the performance metric of synaptic operations per sec (SOPS), or Giga SOPS (GSOPS) as introduced in [22]. One synaptic operation involves reading of an 8-bit synaptic weight and updating the respective post-synaptic neuronal membrane potential. For instance, if

the memory clock rate is 100 MHz, in the case when all the inputs spike, we can read $256/8 = 32$ synaptic weights from each row of the the NVM array and correspondingly update the membrane potentials every 10 ns, thus, giving us 3.2 GSOPS. Table 5.6 lists the different design metrics for the three different crossbar-based designs. We also present the normalized metrics with respect to power (GSOPS/W), area (GSOPS/mm^2), and both area and power (GSOPS/W/mm^2). The metrics are evaluated for a neuro-synaptic core designed with each of the memory cells at 65 nm technology node.

As can be seen, the STT-RAM core has nearly 2× and 5× higher throughput per unit Watt compared to PCM and RRAM based neuro-synaptic cores, respectively. Hence, we designed a larger sized core with STT-RAM bit cells targeted to solve the MNIST classification problem.

5.3.4 STT-RAM NVM Array for BASNN

The MLP SNN for the MNIST problem reported in [46], has four layers as - $784 \times 256 \times 256 \times 10$, which achieves a test accuracy of 98.0% with 8-bit fixed precision representation. Their work also demonstrates a hardware design, with SRAM memory, for this network. In our work, we demonstrate the STT-RAM crossbar-based design for a single layer of this network with 256 neurons. The design can be scaled up for deeper networks by tiling multiple cores as shown in Figure 5.13. Our neuro-synaptic core for SNN inference consists of a memory array of 2048×2048 STT-RAM bit-cells with their corresponding read and write peripheral circuits. From interconnect parameters for the 28 nm node [229], we estimate the resistance of bit-line with 2048 bit cells as $R_{BL} = 4.1\,\text{k}\Omega$ and capacitance as $C_{BL} = 63\,\text{fF}$. Thus, the RC wire delay of about 260 ps is considerably less than the pulse width needed to read or write (2 to 10 ns) to a single STT-RAM device, making our large array design feasible. Every computational core also comprises of an Address Event based spike

decoder, the digital logic for binary activation spiking neuron and a spike router. The on-chip routing network manages the communication of binary spike signals issued by spiking neurons to their respective destination cores, similar to the spike routing architecture in IBM's TrueNorth processor [22].

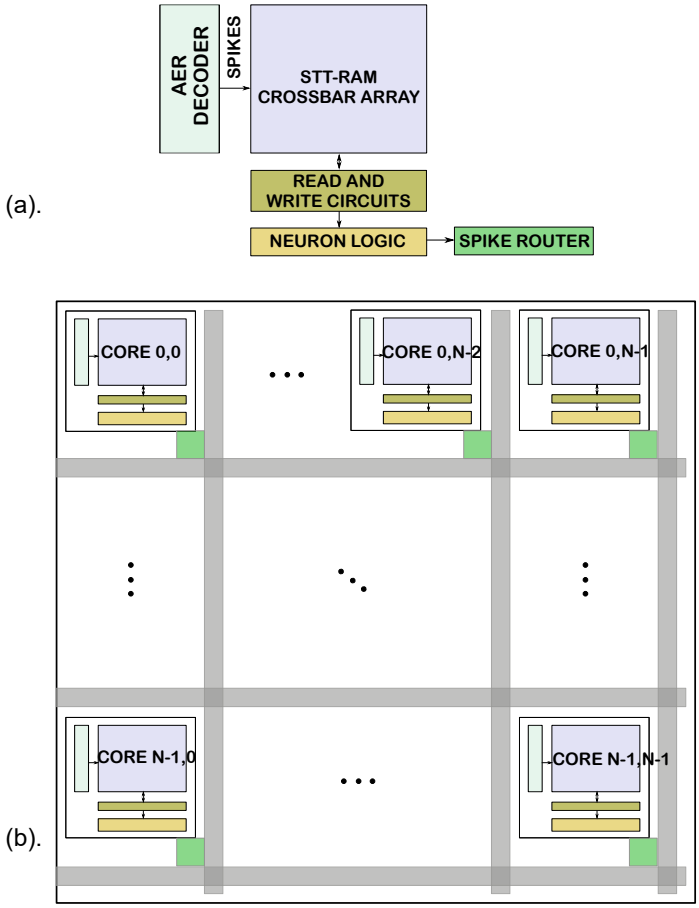

Figure 5.13 (a). Single neuro-synaptic core and its five main components. (b). Multiple cores tiled together for realizing large SNNs. The on-chip router communicates binary spikes to different cores of the chip. The address decoder translates the received spike information into binary spikes for the corresponding wordlines of the crossbar array.

We designed our STT-RAM array along with its the peripheral circuits and digital logic in 28 nm CMOS technology node (with $F = 50\,\text{nm}$). Table 5.7 lists the post-synthesis area and power numbers for the memory and digital logic components of the crossbar array. Using these numbers, we estimate the performance per Watt and per mm^2 for our neuro-synaptic core. Based on the read latency of our STT-RAM

memory block, we use a clock frequency of 100 MHz to read the synaptic weights and biases from the array and operate the neuronal digital blocks. The area and power of the on-chip router for our neuro-synaptic core are taken to be 10% of the total neuronal area and power respectively based on some of the earlier reported designs [22, 198, 230]. The array efficiency of our STT-RAM crossbar array is 53%, with the read/write peripheral circuits taking up 0.27 mm^2 and the 4 Mb of STT-RAM bit-cell array requiring 0.3 mm^2 of the total memory area.

Table 5.7 Post-synthesis Numbers for the STT-RAM-based Neuro-synaptic Core

Blocks	Area (mm^2)	Power (mW)
Digital Logic	0.04	2.82
STT-RAM array	0.57	8.58
Total	0.61	11.40

To benchmark the performance of the STT-RAM based neuro-synaptic core with a full CMOS implementation, we study a slightly modified core with 256 neurons and 1156×256 synapses, as was used in the first layer of the BASNN design reported in [46]. We compare the two designs in terms of energy spent per time-step and the area for realizing a single layer of BASNN in Figure 5.14. While the neuronal energy is not very different in the two designs, the memory read energy for the STT-RAM array is nearly 3× smaller than that of the SRAM memory. Thus, there is a clear energy advantage in using STT-RAM memory over SRAM for neural network inference engine designs, as has also been previously demonstrated for memory cache designs [225]. The small difference in the digital neuronal energy can be attributed to the extra glue and control logic in the SRAM design [46]. Our design does not need the additional glue logic as every neuron in the layer is connected to the bitline or source line of the STT-RAM core as seen in Figure 5.11. Overall, there is a 2.8× reduction in the energy consumption per time-step in our neurosynaptic core

compared to that of the non-crossbar SRAM design (see Figure 5.14(a)). The area advantage of STT-RAM can be seen in Figure 5.14 (b) where we see a 3× reduction in the area for a single SNN layer between the two designs.

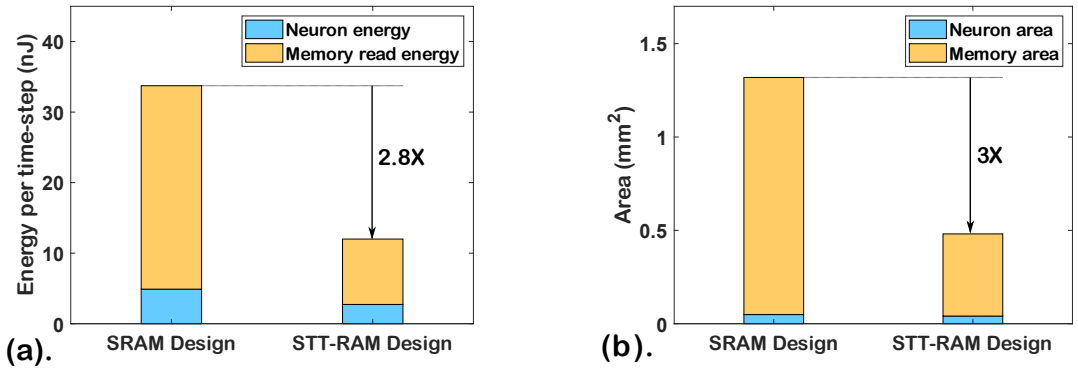

Figure 5.14 Performance comparison between an SRAM based design and STT-RAM based design for 256 neurons and 1156 × 256 synapses. (a). Comparison of the neuronal and memory read energies for a time-step in the SRAM (described in Yin *et al.*, 2017) and STT-RAM designs. The additional glue logic in the SRAM based design results in slightly higher power for the neurons, while such circuitry is not required in the STT-RAM crossbar array, as the neurons directly connect to the synaptic array as in Figure 5.11. (b). Comparison of the neuronal logic area and the memory area between the two designs for a single layer of 256 neurons.

In our design, one synaptic operation is the access of an 8-bit synaptic weight from the memory and updating the membrane potential, when an input spike is received. Table 5.8 shows the different performance metrics for our neuro-synaptic core with the SRAM-based design in [46]. Each spike in the input layer of the core results in accessing 2048 bits of synapses from the array (or accessing 256 synaptic weights) and further performing 256 neuronal updates. Hence, at a clock rate of 100 MHz, our design achieves 25.6 GSOPS.

The SRAM based BASNN design realizes a four-layer MLP network and operates at 163 MHz [46]. Using the spike statistics through the complete network, the design achieved 77.5 GSOPS. As discussed in the previous section, the energy of a single layer of the network per time-step is almost two times smaller for the STT-RAM core (Figure 5.14). On normalizing the total operations per second in each of the two

designs with respect to the total power required per time-step, we see close to 2× improvement in the GSOPS/W for the STT-RAM core (Table 5.8). It was also seen in the previous section that the core area was smaller by a factor of 3 with respect to the SRAM design. Similarly, normalizing the designs' throughput with respect to the total power and area, the GSOPS/W/mm^2 metric for the STT-RAM design is higher by approximately 6× compared to the SRAM based design.

Table 5.8 Performance Comparison Between SRAM and STT-RAM Architectures

Hardware accelerators	GSOPS/W	GSOPS/W/mm^2
SRAM [46] 28 nm	1020	627.5
STT-RAM crossbar (this work) 28 nm	2245	3680

With the assumed dimensions of the memory cell, the STT-RAM design has a density of approximately 0.8 MB/mm^2. At 28 nm node (F=50 nm), the highest memory density achievable using conventional lithography for a 1-bit per cell cross-point technology is 12.5MB/mm^2. Recent work on memristor based inference engines which has reported 800 GSOPS/Watt and 580 GSOPS/mm^2 assumes a 3D memory technology that has a density of 130 MB/mm^2 at 32 nm node [31]. Similarly, the inference engine based on RRAM which has reported 1060 GSOPS/Watt and 820 GSOPS/mm^2 assumes a memory technology that has a density of 54 MB/mm^2 at 32 nm node [28]. Our design is competitive with these studies in terms of performance per Watt.

While the designs reported in the literature [28, 30, 31, 205] make use of memristors' analog conductance states for storing the data, they require the use of ADCs and DACs, bringing down the area efficiency. Our design, on the other hand, uses binary STT-RAM states and avoids the use of ADCs and DACs, giving a balanced memory area efficiency of 53%.

5.3.5 Learning on NVM Array

We extended the above described NVM array to support on-chip spike based learning. As described in the original work, the BASNN training problem is cast into the same gradient based back-propagation framework as that exists for ANNs [46]. The straight through estimator linearizes the sharp jump in the activation output near the threshold to compute the membrane potential's (evaluated as in Equation 5.9) gradient with respect to the synaptic weights. The activation gradient $g^k(n)$ for the k^{th} layer is given as,

$$a^{k\prime}(v^k(n)) = g^k(n) = \begin{cases} (1/2\theta), (\theta - 1) \leq v^k(n) \leq (\theta + 1) \\ 0, \ \textit{otherwise} \end{cases} \tag{5.11}$$

The gradient based back-propagation requires evaluating the gradients of the loss function ($\mathcal{L}(\mathbf{w})$) with respect to the network parameters ($\mathbf{w}$ and $\mathbf{b}$) using the chain rule for derivatives. Finally, the weight update term which is proportional to the loss gradient ($\eta \partial \mathcal{L} / \partial \mathbf{w}^k$) is added to the current set of weights for each layer. The amount of weight update is decided by the learning rate, η. This process involves the following set of expressions to be evaluated. The loss function $\mathcal{L}$ adopted for training is the squared hinge loss [46]. Weight update for each layer k, is given as,

$$\Delta \mathbf{w}^k = \eta \boldsymbol{\delta}^k \times (\mathbf{a}^{k-1})^T \tag{5.12}$$

Here, $\mathbf{a}^{k-1}$ is the vector of spikes output from the previous layer $k-1$. The error gradient is represented by the term $\boldsymbol{\delta}^k$, which is iteratively calculated for each layer starting from the last layer $L = k + 1$, as,

$$\boldsymbol{\delta}^k = \left((\boldsymbol{w}^{k+1})^T \boldsymbol{\delta}^{k+1} \right) \circ \boldsymbol{a}^{k\prime} \tag{5.13}$$

For the last layer, L, the gradient term is evaluated as,

$$\boldsymbol{\delta}^L = \mathcal{L}' \circ (-\boldsymbol{S}^d) \tag{5.14}$$

Here, $\boldsymbol{S}^d$ is the desired set of spike trains at the output layer of the network.

While we used a 8-bit fixed precision representation for network parameters and forward pass computation for inference, training requires all the network variables to have a larger dynamic range varying over 5 to 6 orders of magnitude [219, 220]. Our studies on the dynamic range of weights and gradients while training the BASNN showed that the values ranged between 10^{-5} to 1 making the floating point representation is an ideal choice to represent variables with such large dynamic range [219].

The network we carried out our studies is the same as described in the previous section, which has an MLP configuration. The input layer with 784 pixels connects in an all-to-all manner with hidden layer of 256 neurons. There are two hidden layers each with 256 neurons and finally the output layer has 10 neurons. We trained this network with a batch-size of 1 and stochastic gradient descent (SGD) optimizer.

For the baseline results, we used the single precision representation of 32-bit floating point (also referred to as FP-32). This representation uses 8-bit for exponent, 23-bits for mantissa and 1-bit for the sign [231]. The four-layered SNN was trained for 500 epochs on the MNIST dataset and achieved an accuracy of 97.25% on the test set, and 98.0% on the training set. We also introduced a dropout of 0.1 in the input layer and 0.2 in the hidden layers. Note that here, we employed the stochastic gradient descent with a batch size of 1 to avoid the need to store the intermediate variables of the size of the network parameters outside of the array. The original work on this algorithm with a 4 layered network and with hidden layers of size 1024 in the FP-32 representation reports a test accuracy of 98.7% [46] which was carried out by making use of the Adam optimizer and using a batch size of 100. To emulate the reduced precision on the hardware we carried out the training in half precision floating-point format (also referred to as binary16 or FP-16) in MATLAB. This representation uses IEEE-754 2008 format with 5-bits for exponent and 10-bits for the mantissa and can

provide a dynamic range of 10^5 for the representation [232]. The network emulated with FP-16 representation for the variables and computations (and the same training scheme with batch of 1 and SGD), achieved an MNIST test accuracy of 97.18% and training accuracy of 97.8%. Hence, there was no significant drop in the algorithmic performance going to lower precisions.

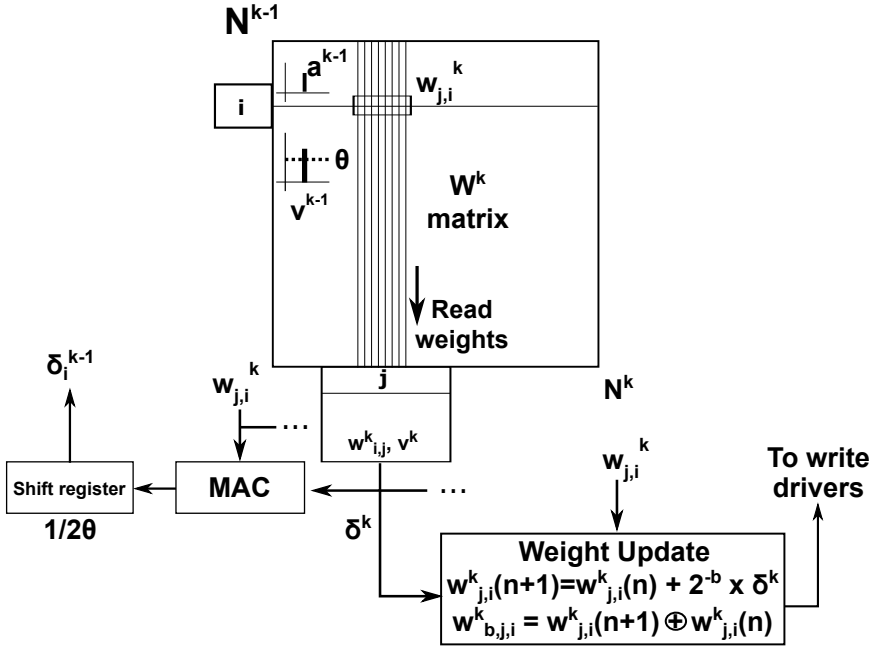

Figure 5.15 Scheme for performing back-propagation of error gradients and weight update.

Figure 5.15 shows the scheme for realizing the learning computations on the crossbar array. It involves two phases, one for evaluating the gradients δ and second for performing the weight updates. The forward pass evaluates the neuronal spike (a^k), membrane potentials (v^k) and the activation gradients (g^k). For each pre-synaptic neuron i, which has issued a spike at time-step n, synaptic weights on the i^{th} row of the crossbar are read and stored in local registers of the post-synaptic neurons. If the pre-synaptic gradient is non-zero (or if $v_i^{k1} \in [0, 2\theta]$), we then compute the δs as the output of a MAC (multiply and accumulate) unit as,

$$\delta_i^{k-1} = \frac{1}{2\theta} \sum_j w_{j,i}^k . \delta_j^k \tag{5.15}$$

The weight update term $\Delta \boldsymbol{w}$ is evaluated only if there is a pre-synaptic spike, i.e., if $a_i^{k-1} \neq 0$. We use the $w_{j,i}$s that were read in the previous phase (or read again, if they were not read earlier) and add the update term to each of the synaptic weights.

$$w_{j,i}(n+1) = w_{j,i}(n) + \Delta w_{j,i} \tag{5.16}$$

The bits flipped from the read weight values and the newly updated values are compared by performing a bit-wise XOR as, $\boldsymbol{w}_{flip} = \boldsymbol{w}(n+1) \oplus \boldsymbol{w}(n)$. The two steps of weight update and evaluating δs are repeated sequentially over all the wordlines which have received an input spike.

While the array design for inference had 8-bits to represent the synaptic weights, we used 16-bits for training. Figure 5.16 shows the modified configuration for the 2048×2048 sized NVM array. The array now directly interfaces with 128 post-synaptic neurons at a time, thus giving a fan-out of 128 for every input. For larger fan-outs, a single input spike can be used to activate multiple wordlines in a time-multiplexed manner. The digital logic blocks at the array periphery consists of 16-bit floating point adder, FP-16 MAC unit, and an XOR block to identify the flipped bits in the weights and biases. Additionally, it also includes a set of schedulers to select the different wordlines for a given input spike.

We designed the digital logic with FP-16 compute units in Verilog and synthesized them using the Synopsys Design Compiler tool to get the area and power estimates of each of these blocks at 65 nm node. The memory array design remains the same as in Section 5.3.4 with 2048 × 2048 bit cells with a memory capacity of 512 KB. Each synapse is represented with 16 bit-cells, which also implies every read cycle we can access 128 synaptic weights on a row. Table 5.9 presents the synthesized area and power estimates for the different digital logic blocks (FP-16 neuron, MAC unit, spike decoder, router and controller). The area and power for the spike router is taken as 10% of that the neuronal design based on previous published works [230].

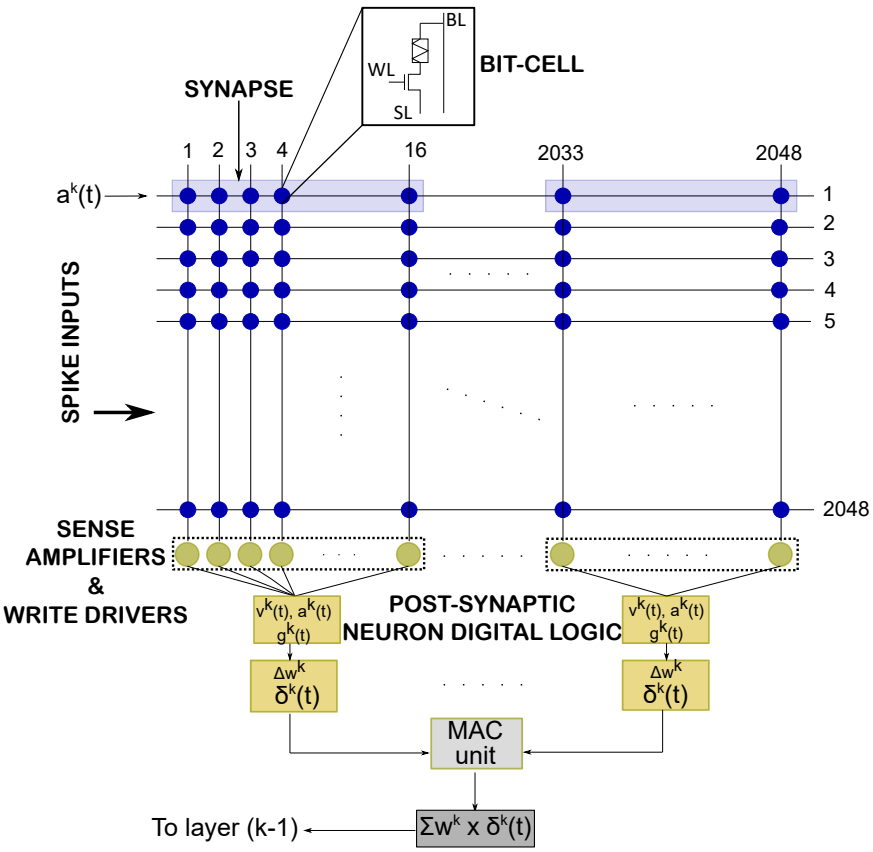

Figure 5.16 A 2048 × 2048 crossbar array supporting access to 2048 inputs and 128 outputs at a time. Each synaptic weight is represented by 16 devices on a row. The peripheral digital logic consists of blocks to update the neuron membrane potential, the error derivative δ and the weight update terms Δw. Similar to the design for inference (Figure 5.13), when multiple cores are tiled together, the inter-core spike communication takes place through the routers and spikes are presented to the memory array via address decoders. To support fan-out larger than 128, multiple wordlines can be accessed in a time-multiplexed manner for every batch of 128 post-synaptic neuron.

We compare our design with an SRAM memory block of the same capacity as the STT-RAM array of 512 KB. The DESTINY tool is used to get the estimates for the SRAM memory block [233]. Similar to the SRAM cell used in the inference design, we use a bit cell of size $150F^2$ in DESTINY. Table 5.10 presents the read and write peripheral power for the two memory technologies of SRAM and STT-RAM. Based on the maximum latency of SRAM access and bandwidth reported by DESTINY, we set the memory operating clock frequency at 250 MHz. The STT-RAM array that we designed can operate at 100 MHz based on the timing requirements of its peripheral

circuits. The total area of the STT-RAM design is 1.83 mm^2 and that of the SRAM design is 7.27 mm^2.

Table 5.9 Post-synthesis Numbers for Floating-Point Digital Logic Blocks

Blocks	Area (mm^2)	Power (mW)
Neuron Logic (Forward pass)	0.358	20.49
Spike Router	0.036	2.05
Controller and decoder	0.11	6.15
MAC and weight update blocks	0.039	18.21
Total	0.543	46.9

Table 5.10 Comparison of SRAM and STT-RAM Architectures for Training

Design Parameters	SRAM (DESTINY)	STT-RAM
Operating Frequency (MHz)	270	100
Read Power (mW)	529.5	27.29
Write Power (mW)	555.25	760.19
Memory bit-cell area (mm^2)	6.29	1.22
Peripheral area (mm^2)	0.44	0.17

Our digital logic blocks synthesized with 65 nm library cells can operate at 500 MHz, without incurring any timing violation. The arrival of a spike on the input wordline enables the read of all the bit cells on a row in the memory array. For the STT-RAM array, the logic (running at 500 MHz) operates 5× faster than the memory (operating at 100 MHz), while for the SRAM memory (operating at 250 MHz), the logic clock is 2× faster than memory. The neuronal updates, as well as the evaluations of δ and Δw values take place in the digital logic. The back-propagation modules used for computing δs perform multiplication of δ_i^k and $w_{i,j}^k$ across all the output neurons i, of layer k, and hence need more clock cycles than the memory read/write per row j.

The total cycles needed for writing to the weights through write drivers and computing error back-propagation $\boldsymbol{\delta}^{k-1}$ is decided based on the maximum of the cycles needed to write to the memory array and to perform the MAC operation as the two stages can be parallelized. For the STT-RAM array, we make use of only two write drivers per 16-bit synapse, to limit the STT-RAM write driver power requirement and hence, each write in the STT-RAM array requires 8 memory clock cycles. To measure the performance, we compute the synaptic operations during the forward pass, back-propagation, and weight update, based on the spike and gradient statistics collected while training the network in software emulation with half-precision representation. Our neurosynaptic core can be used to realize one layer of the SNN with maximum of 128 neurons. For realizing layers with 256 neurons, two of such cores can be interfaced to the previous layer output.

We measure the performance of a single core, by measuring the time required to perform the synaptic reads for neuronal potential update (forward pass), synaptic reads for evaluating the MAC (for δ) and finally the synaptic reads and writes during the weight update stage (using Δw). Table 5.11 lists the average number of synaptic operations during different stages of learning in the neurosynaptic core. Using the listed statistics we estimate the GSOPS (Giga Synaptic Operations per second) for the neurosynaptic core. Table 5.12 presents the performance comparison of the SRAM and STT-RAM designs. While the SRAM can be operated at a higher frequency than the STT-RAM, the overall throughput is limited by the MAC logic which takes more cycles per memory access, hence, we do not see a significant difference in the GSOPS in the two designs. As we have accounted for the network activity during training, the number of synaptic writes is significantly smaller than the number of synaptic reads (in this example, we have $1,180$ writes compared to $8,224$ reads in both forward and backward passes per image). This translates to smaller overall energy requirement for the STT-RAM core, as STT-RAM memory read energy is significantly less than

Table 5.11 Average Spike Statistics per Layer per Core in the SNN during Training

Network Statistics	**SRAM (250 MHz)**	**STT-RAM (100 MHz)**
Incoming Spikes $\mathbf{a}^{k-1}$	20	20
Incoming gradients $\mathbf{g}^{k-1}$	95	95
Forward Pass		
No. of synaptic reads	2560	2560
Cycles for read	20	20
Back-propagation		
No. of synaptic reads	5664	5664
Cycles for MAC	2833	1152
Weight Update		
No. of writes	1180	1180
Cycles for write	20	160
Overall GSOPS		
Synaptic Operations (SOPs)	9404	9404
Total cycles	2853	1172
Cycle time (us)	0.004	0.01
GSOPS	0.82	0.80

that of the SRAM memory (by $\sim$7$\times$). Hence, considering the total power in the design for the forward and backward passes, the STT-RAM core has $\sim$5$\times$ higher GSOPS/W compared to that of the SRAM core. Normalizing the performance with respect to the core area, it can be seen that STT-RAM core has nearly 20$\times$ higher GSOPS/W/mm^2 than SRAM core.

Table 5.12 Performance Comparison Between SRAM and STT-RAM Designs

Design	GSOPS	GSOPS/W	GSOPS/W/mm^2
SRAM design	0.82	0.71	0.09
STT-RAM	0.80	3.5	1.93

5.4 Summary and Discussion

We have presented a an scalable CMOS architecture which could be extended in designing larger spike based learning systems. The supervised learning algorithm ReSuMe has been analyzed in terms of its learning speed and the network size required. The results presented here give an indication of the size of the network to be used for longer spike streams. It was seen that an on-off ratio of just 100 is sufficient to represent the weights, with a precision of 5 bits. Based on this analysis, we have presented a high level scalable architecture for on-chip learning, inspired by the recently demonstrated SNN implementations. At an acceleration of 1000 our design is projected to scale to support 30 MSUPS/Watt at 10 nm node.

We also presented a convolutional neural network in both spiking (SNN) and non-spiking (ANN) versions, realized using memristive cross-bar arrays. The networks are trained in software in the full-precision mode and their synaptic weights are then optimized for designing an inference engine using memristive devices. Our simulations suggest that optimization strategies such as weight clipping and realization of the convolution operation in parallel using memristive arrays can result in implementations with close-to-baseline accuracies. As our SNN performs nearly as well as the reference ANN on memristive hardware with conductance variability, it shows potential for realizing energy efficient neuromorphic platforms using SNNs.

We have also presented a scalable crossbar based design for accelerating SNNs, using binary storage NVM arrays. We compared the designs for SNN inference across three prominent NVM technologies, STT-RAM, RRAM and PCM. With the STT-

RAM array showing a higher throughput compared to the other two, we designed a 2048×2048 sized array to realize the BASNN model. This design is able to achieve nearly twice the performance per Watt as compared to a full CMOS design with SRAM memory. Comparing the performance per unit area and per unit Watt metric (GSOPS/W/mm^2), our neurosynaptic core for inference is almost six times better than SRAM based CMOS design.

We extended the proposed crossbar architecture to incorporate learning on the hardware. We have also evaluated the BASNN training algorithm under reduced bit-precision, with less than 1% drop in the accuracy from the 32-bit floating point baseline. Overall, the STT-RAM core performs nearly 20$\times$ better than the equivalent SRAM core, in terms of GSOPS/W/mm^2, due to its lower read energy and smaller bit-cell area, when considering the network statistics for training with the MNIST dataset. Our design's throughput is limited by the number of cycles required by the MAC unit. Introducing more parallelization in the MAC logic can improve the overall performance of the design, which could be a future direction to this work.

Going forward, the network training under further low bit-precisions (≤ 8 bits) could be studied and hardware-aware optimization strategies during training to get the best accuracy could be explored. Another aspect that is worth exploring is a comparison between other emerging nanoscale memory devices such as PCMO, FeRAM, etc. and some of the existing CMOS based NVMs such as Flash memory technology, for designing efficient neural network accelerators.

CHAPTER 6

CONCLUSION AND FUTURE OUTLOOK

We have explored the algorithmic and hardware aspects of bio-inspired computing with SNNs in this dissertation. On the algorithms front, we have explored the spike-based supervised rules of ReSuMe and NormAD, which are capable of training spiking neural networks to generate spikes at desired instants of time. A more hardware-friendly algorithm with integrate and fire neuron model, called the binary activation SNN is also explored and a non-von Neumann design is presented for SNN inference and supervised learning.

We have developed a network for handwritten digit classification trained using the spike based supervised learning algorithm NormAD. The two-layered network, achieves a classification accuracy of 98.17% on the standard data-sets of handwritten digits, the MNIST database. We achieve this accuracy with less than 4× fewer parameters than the state of the art networks, with just $\sim 2\%$ drop in accuracy. These include optimizing the two layer network discussed in this work, in terms of number of learning synapses, bit-precision and other algorithmic improvements for efficient hardware implementation without any reduction in the classification accuracy. All of our NormAD SNN simulations are carried out on the GPU and we have also extended this to demonstrate real-time prediction on the users' hand-drawn digits on a touch-screen.

Devices such as PCMs, STT-RAMs, and RRAMs have shown potential to be incorporated in neuromorphic computing hardware architectures [113, 198]. We have developed an efficient hardware architecture that is capable of executing online on-chip learning for different cognitive tasks. The system is designed using the emerging NVM devices to store the synaptic weights and CMOS circuits for neuronal

computation. We also developed computationally efficient models of three nanoscale devices which capture the basic features of memory switching and incorporate the reliability aspects as well. These modeling scheme presented could be extended to more emerging devices or fine-tuned based on further improvements in device engineering techniques, especially at advanced nodes.

We have shown memristive hardware for SNN acceleration with both analog and digital storage arrays. While analog memory arrays are area efficient and also provide potentially higher throughput, there are several reliability challenges, which need to be addressed either at the algorithmic or architecture level. We have also presented an NVM based architecture for realizing SNN inference and performing training on the hardware. We have shown that the STT-RAM technology performs better than conventional SRAM memory which is typically used in on-chip caches and registers. Overall, the non-volatility combined with smaller footprint of the NVM devices could potentially replace the conventional volatile memories in edge devices which may be having intermittent supply of power.

Going forward, we consider time-series based problems where SNNs have the potential to perform better than other neural network models, as an application domain worth further research due to their inherent temporal processing capability [234–236]. Another aspect is the study on scalability and generalizability of such non-von Neumann architecture to larger class of problems. This is essential to identify the scale of applications whose computations can be performed entirely within these NVM based accelerators without the dependence on communicating with the cloud servers.

APPENDIX A

SIMULATION PARAMETERS

The simulation parameters for the spike based learning algorithm NormAD used in Chapter 3 are listed in Table A.1. Table A.2 lists the simulation parameters and configurations for the Remote Supervised Method (ReSuMe) that were used in the CMOS digital hardware design detailed in Chapter 5.

Table A.1 Simulation Parameters for NormAD

Parameters	**Values**
time step Δt	0.1 ms
Membrane conductance, g_L	30 nS
Resting potential, E_L	-70 mV
Threshold potential, V_T	20 mV
Learning rate (12 kernels), r	26 pS
Learning rate (8 kernels), r	31 pS

Table A.2 Simulation Parameters for ReSuMe

Parameters	**Values**
Common parameters	
time step Δt	0.1 ms
Membrane conductance, g_L	1μ S
Resting potential, E_L	-62 mV
Threshold potential, V_{th}	-55 mV
Learning window, amplitude A_+	1×10^{-10}
Learning window, time constant, τ_+	20 ms
Non-Hebbain term, a_0, a_d	0
NMC neurons	
Membrane capacitance, C	0.2 μF
Synaptic time constant, τ	6 ms
Synaptic weight multiplier, w_0,	5×10^{-8}
Output neuron	
Membrane capacitance, C	1 nF
Synaptic time constant, τ	1.5 ms
Synaptic weight multiplier, w_0,	$1 \times 10^{-9.5}$
Network parameters	
No. of NMC neurons	800 (typical case)
Input connectivity to NMC	30%
NMC neurons to output connectivity	70%
Excitatory to inhibitory neurons in NMC	80 : 20

APPENDIX B

NVM DEVICE MODEL PARAMETERS

The parameters used in the compact Verilog-A models of PCM, STT-RAM and RRAM are listed here that were discussed in Chapter 4.

Table B.1 Parameters used in the PCM Compact Model

Parameter	Value	Parameter	Value
R_{th}	65 MK/W	R_{s1}	1 $\mu\Omega$
C_1	1 nF	R_{s2}	1 Ω
R_{t1}	1 GΩ	R_{heater}	410 Ω
R_{t2}	1 $\mu\Omega$	E_a (in eV)	$0.27 - 0.25c_x$
T_X	200°C	Γ_a	2.3 eV
T_M	600°C	A	1.06×10^{-23} ns
C_2	20 nF	V_h	0.5 V
R_{ON}	1000 Ω	I_{th}	40 μA

Table B.2 Parameters used in the STT-RAM Compact Model

Parameter	Value	Parameter	Value
R_P	3 kΩ	H	1.5×10^5 A/m
R_{AP}	6 kΩ	γ	1.76×10^{11} rad/s.T
I_{write}	150 μA	M_S	4.65×10^5 A/m
α	0.02	K	1.15×10^5 T.A/m
η_1 (HRS)	0.42	η_2 (LRS)	0.17

Table B.3 Parameters used in the RRAM Compact Model

Parameter	Value	Parameter	Value
R_P	$60\,\mathrm{k\Omega}$	γ_0	16
R_{AP}	$500\,\mathrm{k\Omega}$	E_a	0.60
g_0	0.75	t_{ox}	$12\,\mathrm{nm}$
α	0.02	g_{max}	$1.7\,\mathrm{nm}$
V_0	0.75	g_{min}	$0.1\,\mathrm{nm}$

BIBLIOGRAPHY

[1] M. Fischetti, "Brain versus Computers," 2011, viewed on August 20, 2019. Available at https://www.scientificamerican.com/article/computers-vs-brains/.

[2] M. L. McCorkle, "ORNL Launches Summit Supercomputer," 2018, viewed on September 17, 2019. Available at https://www.ornl.gov/news/ornl-launches-summit-supercomputer.

[3] N. A. of engineering, "Grand Challenges - Reverse-engineer the Brain," 2009, viewed on October 7, 2019. Available at http://bit.ly/1PmsLiX.

[4] E. R. Kandel, J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. Hudspeth, *Principles of Neural Science*. New York, NY:McGraw-hill, 2000, vol. 4.

[5] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[7] J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 3642–3649.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[9] Y. Goldberg, "A Primer on Neural Network Models for Natural Language Processing." *Journal of Artificial Intelligence Research (JAIR)*, vol. 57, pp. 345–420, 2016.

[10] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[11] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale Video Classification with Convolutional Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1725–1732.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[13] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1319–1327.

[14] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[15] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 173–182.

[16] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[17] A. van Schaik and Shih-Chii Liu, "AER EAR: A matched silicon cochlea pair with address event representation interface," in *IEEE International Symposium on Circuits and Systems*, May 2005, pp. 4213–4216 Vol. 5.

[18] J. White, T. A. Dickinson, D. R. Walt, and J. S. Kauer, "An olfactory neuronal network for vapor recognition in an artificial nose," *Biological Cybernetics*, vol. 78, no. 4, pp. 245–251, 1998.

[19] T. Horiuchi and K. Hynna, "Spike-based VLSI modeling of the ILD system in the echolocating bat," *Neural Networks*, vol. 14, no. 6-7, pp. 755–762, 2001.

[20] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Hafliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, 2011.

[21] B. Wang, W. Ke, J. Guang, G. Chen, L. Yin, S. Deng, Q. He, Y. Liu, T. He, R. Zheng, Y. Jiang, X. Zhang, T. Li, G. Luan, H. D. Lu, M. Zhang, X. Zhang, and Y. Shu, "Firing frequency maxima of fast-spiking neurons in human, monkey, and mouse neocortex," *Frontiers in Cellular Neuroscience*, vol. 10, p. 239, 2016.

[22] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[23] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[24] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," *arXiv preprint arXiv:1812.01739*, 2018.

[25] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, p. 106, 2019.

[26] T. Gokmen and Y. Vlasov, "Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations," *Frontiers in Neuroscience*, vol. 10, p. 333, Jul 2016.

[27] S. Kim, T. Gokmen, H. Lee, and W. E. Haensch, "Analog CMOS-based resistive processing unit for deep neural network training," in *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 422–425.

[28] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[29] L. Song *et al.*, "PipeLayer: A pipelined ReRAM-Based Accelerator for Deep Learning," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 541–552.

[30] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, vol. 44, no. 3, 2016, pp. 27–39.

[31] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: ACM, 2019, pp. 715–731.

[32] H. Yan, H. R. Cherian, E. C. Ahn, and L. Duan, "CELIA: A device and Architecture Co-Design Framework for STT-MRAM-Based Deep Learning Acceleration," in *International Conference on Supercomputing*, ser. ICS '18. New York, NY, USA: ACM, 2018, pp. 149–159.

[33] B. Sun, D. Liu, L. Yu, J. Li, H. Liu, W. Zhang, and T. Torng, "MRAM co-designed processing-in-memory CNN accelerator for mobile and IoT Applications," *arXiv preprint arXiv:1811.12179*, 2018.

[34] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "TIME: A Training-in-memory Architecture for RRAM-based Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[35] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[36] S. R. Kulkarni, A. V. Babu, and B. Rajendran, "Acceleration of convolutional networks using nanoscale memristive devices," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2018, pp. 240–251.

[37] N. Anwani and B. Rajendran, "NormAD-Normalized Approximate Descent based supervised learning rule for spiking neurons," in *International Joint Conference on Neural Networks*, July 2015, pp. 1–8.

[38] F. Ponulak and A. Kasinski, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[39] J. Hawkins and S. Blakeslee, *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[41] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, pp. 4278–4284.

[42] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[43] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[44] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proceedings of the National Academy of Sciences*, vol. 105, no. 9, pp. 3593–3598, 2008.

[45] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 99–115, 1990.

[46] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J. Seo, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," in *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct 2017, pp. 1–5.

[47] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[48] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2961–2969.

[49] Y. LeCun, C. Cortes, and J. C. C. Burges, "The MNIST database of handwritten digits," 1998, viewed on May 1, 2015. Available at http://yann.lecun.com/exdb/mnist/.

[50] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[51] I. Wallach, M. Dzamba, and A. Heifets, "Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery," *arXiv preprint arXiv:1510.02855*, 2015.

[52] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[53] R. Gutig and H. Sompolinsky, "The tempotron: a neuron that learns spike timing-based decisions," *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, Mar 2006.

[54] J. M. Brader, W. Senn, and S. Fusi, "Learning Real-World Stimuli in a Neural Network with Spike-Driven Synaptic Dynamics," *Neural Computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

[55] J. J. Hopfield and C. D. Brody, "Learning rules and network repair in spike-timing-based computation networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. 1, pp. 337–342, 2004.

[56] S. Santurkar and B. Rajendran, "C. elegans chemotaxis inspired neuromorphic circuit for contour tracking and obstacle avoidance," in *International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[57] A. Bora, A. Rao, and B. Rajendran, "Mimicking the worm - An adaptive spiking neural circuit for contour tracking inspired by C. Elegans thermotaxis," in *International Joint Conference on Neural Networks (IJCNN)*, July 2014, pp. 2079–2086.

[58] C. Shetty, S. Nitchith, R. Rawat, S. R. Nandakumar, P. Shah, S. Kulkarni, and B. Rajendran, "Live demonstration: Spiking neural circuit based navigation inspired by C. elegans thermotaxis," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1905–1905.

[59] F. Ponulak, "Supervised learning in spiking neural networks with resume method," *Phd, Poznan University of Technology*, vol. 46, p. 47, 2006.

[60] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks using Backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.

[61] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, 2015.

[62] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[63] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell, "The action potential and conduction of electric impulses," in *Molecular Cell Biology. 4th edition.* WH Freeman, 2000.

[64] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Journal of Physiology*, vol. 117, no. 4, pp. 500–544, Aug 1952.

[65] H. Paugam-Moisy and S. Bohte, "Computing with spiking neuron networks," *Handbook of natural computing*, pp. 335–376, 2012.

[66] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical Journal*, vol. 35, no. 1, pp. 193–213, Jul 1981.

[67] R. FitzHugh, "Impulses and Physiological States in Theoretical Models of Nerve Membrane," *Biophysical Journal*, vol. 1, pp. 445–466, Jul 1961.

[68] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[69] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of Neurophysiology*, vol. 94, pp. 3637–3642, Nov 2005.

[70] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, pp. 303–304, 1999.

[71] W. Gerstner, "Chapter 12 a framework for spiking neuron models: The spike response model," in *Neuro-Informatics and Neural Modelling*, ser. Handbook of Biological Physics, F. Moss and S. Gielen, Eds. North-Holland, 2001, vol. 4, pp. 469 – 516.

[72] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[73] D. Hebb, *Organization of Behavior*. New York, NY: Wiley and Sons, 1949.

[74] G. Q. Bi and M. M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, Dec 1998.

[75] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, Feb. 2010.

[76] A. Tavanaei and A. S. Maida, "Multi-layer unsupervised learning in a spiking convolutional neural network," in *International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2023–2030.

[77] N. Rathi, P. Panda, and K. Roy, "STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668–677, 2018.

[78] T. Masquelier and S. J. Thorpe, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLOS Computational Biology*, vol. 3, no. 2, pp. 1–11, Feb 2007.

[79] P. Panda and K. Roy, "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *International Joint Conference on Neural Networks*, July 2016, pp. 299–306.

[80] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, 2017.

[81] S. Roy and A. Basu, "An Online Unsupervised Structural Plasticity Algorithm for Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 900–910, April 2017.

[82] J. M. Allred and K. Roy, "Unsupervised incremental STDP learning using forced firing of dormant or idle neurons," in *International Joint Conference on Neural Networks (IJCNN)*, July 2016.

[83] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[84] A. Mohemmed *et al.*, "SPAN: Spike Pattern Association Neuron for Learning Spatio-Temporal Spike Patterns," *International Journal of Neural Systems*, vol. 22, no. 04, 2012.

[85] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Precise-Spike-Driven Synaptic Plasticity: Learning Hetero-Association of Spatiotemporal Spike Patterns," *PLOS ONE*, vol. 8, no. 11, pp. 1–16, Nov 2013.

[86] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "DL-ReSuMe: A Delay Learning-Based Remote Supervised Method for Spiking Neurons," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 3137–3149, Dec 2015.

[87] X. Xie, H. Qu, Z. Yi, and J. Kurths, "Efficient Training of Supervised Spiking Neural Network via Accurate Synaptic-Efficiency Adjustment Method," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 6, pp. 1411–1424, June 2017.

[88] J. Tapson, G. Cohen, S. Afshar, K. Stiefel, Y. Buskila, R. Wang, T. J. Hamilton, and A. van Schaik, "Synthesis of neural networks for spatio-temporal spike pattern recognition and processing," *arXiv preprint arXiv:1304.7118*, 2013.

[89] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "CONE: Convex-Optimized-Synaptic Efficacies for Temporally Precise Spike Mapping," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 849–861, April 2017.

[90] C. Lee, S. S. Sarwar, and K. Roy, "Enabling spike-based backpropagation in state-of-the-art deep neural network architectures," *arXiv preprint arXiv:1903.06379*, 2019.

[91] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[92] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *arXiv preprint arXiv:1612.04052*, 2016.

[93] N. K. Kasabov, "NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014.

[94] N. Kasabov, N. M. Scott, E. Tu, S. Marks, N. Sengupta, E. Capecci, M. Othman, M. G. Doborjeh, N. Murli, R. Hartono, J. I. Espinosa-Ramos, L. Zhou, F. B. Alvi *et al.*, "Evolving spatio-temporal data machines based on the NeuCube neuromorphic framework: design methodology and selected applications," *Neural Networks*, vol. 78, pp. 1–14, 2016.

[95] J. Wang, A. Belatreche, L. P. Maguire, and T. M. McGinnity, "SpikeTemp: An Enhanced Rank-Order-Based Learning Approach for Spiking Neural Networks with Adaptive Structure," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 30–43, Jan 2017.

[96] ——, "SpikeComp: An Evolving Spiking Neural Network with Adaptive Compact Structure for Pattern Classification," in *22nd International Conference on Neural Information Processing (ICONIP) - Part II*, S. Arik, T. Huang, W. K. Lai, and Q. Liu, Eds. Cham: Springer International Publishing, 2015, pp. 259–267.

[97] T. Takuya, T. Haruhiko, K. Hiroharu, and T. Shinji, "A training algorithm for spike sequence in spiking neural networks – a discussion on growing network for stable training performance," in *12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, Aug 2016, pp. 1773–1777.

[98] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition – supervised learning and network optimization," *Neural Networks*, vol. 103, pp. 118 – 127, 2018.

[99] S. R. Kulkarni, J. M. Alexiades, and B. Rajendran, "Learning and real-time classification of hand-written digits with spiking neural networks," in *24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2017, pp. 128–131.

[100] S. R. Kulkarni, J. M. Alexiades, and B. Rajendran, "Live demonstration: Image classification using bio-inspired spiking neural networks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–1.

[101] S. Panzeri, N. Brunel, N. K. Logothetis, and C. Kayser, "Sensory neural codes using multiplexed temporal scales," *Trends in Neurosciences*, vol. 33, no. 3, pp. 111 – 120, 2010.

[102] A. A. Lazar, E. K. Simonyi, and L. T. Tóth, "Time encoding of bandlimited signals, an overview," in *Proceedings of Conference on Telecommunication Systems, Modeling and Analysis*. Citeseer, 2005.

[103] J. Wang, A. Belatreche, L. Maguire, and M. McGinnity, "Online versus offline learning for spiking neural networks: A review and new strategies," in *IEEE 9th International Conference on Cybernetic Intelligent Systems*, Sept 2010, pp. 1–6.

[104] S. Q. Khan, A. Ghani, and M. Khurram, "Population coding for neuromorphic hardware," *Neurocomputing*, vol. 239, pp. 153 – 164, 2017.

[105] A. Calderón, S. Roa, and J. Victorino, "Handwritten digit recognition using convolutional neural networks and gabor filters," *International Congress on Computational Intelligence (CIIC)*, 2003.

[106] S. Schreiber, J.-M. Fellous, D. Whitmer, P. Tiesinga, and T. J. Sejnowski, "A new correlation-based measure of spike timing reliability," *Neurocomputing*, vol. 52, pp. 925–931, 2003.

[107] J. Tapson, P. De Chazal, and A. van Schaik, "Explicit computation of input weights in extreme learning machines," in *ELM Volume 1: Algorithms and Theories*, J. Cao, K. Mao, E. Cambria, Z. Man, and K.-A. Toh, Eds. Cham: Springer International Publishing, 2015, pp. 41–49.

[108] L. Wan *et al.*, "Regularization of Neural Networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066.

[109] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Frontiers in Neuroscience*, vol. 9, p. 222, 2015.

[110] S. R. Kulkarni and B. Rajendran, "Scalable digital CMOS Architecture for Spike based Supervised Learning," in *16th International Conference Engineering Applications of Neural Networks (EANN)*. Cham: Springer International Publishing, Sept. 2015, pp. 149–158.

[111] M. Harris, "Optimizing parallel reduction in CUDA," available at http://bit.ly/2gd7fSb.

[112] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to OpenCV," in *35th International Convention MIPRO*, May 2012, pp. 1725–1730.

[113] B. Rajendran and F. Alibart, "Neuromorphic computing based on emerging memory technologies," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 198–211, June 2016.

[114] A. Ankit, A. Sengupta, P. Panda, and K. Roy, "RESPARC: A reconfigurable and energy-efficient architecture with Memristive Crossbars for deep Spiking Neural Networks," in *54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[115] S. Swami and K. Mohanram, "Reliable nonvolatile memories: Techniques and measures," *IEEE Design & Test*, vol. 34, no. 3, pp. 31–41, 2017.

[116] A. V. Babu and B. Rajendran, "Stochastic deep learning in memristive networks," in *24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2017, pp. 214–217.

[117] S. Lee, J.-h. Jeong, T. S. Lee, W. M. Kim, and B.-k. Cheong, "A study on the failure mechanism of a phase-change memory in write/erase cycling," *IEEE Electron Device Letters*, vol. 30, no. 5, pp. 448–450, 2009.

[118] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[119] S. Lai, "Current status of the phase change memory and its future," in *IEEE International Electron Devices Meeting*, Dec 2003, pp. 10.1.1–10.1.4.

[120] A. Chintaluri, H. Naeimi, S. Natarajan, and A. Raychowdhury, "Analysis of defects and variations in embedded spin transfer torque (stt) mram arrays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 3, pp. 319–329, 2016.

[121] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in RRAM computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, March 2018.

[122] T. Wang and J. Roychowdhury, "Well-posed models of memristive devices," *arXiv preprint arXiv:1605.04897*, 2016.

[123] A. G. Mahmutoglu, T. Wang, A. Gupta, and J. Roychowdhury, "Well-posed device models for electrical circuit simulation," 2017.

[124] S. R. Kulkarni, D. V. Kadetotad, J.-s. Seo, and B. Rajendran, "Well-Posed Verilog-A Compact Model for Phase Change Memory," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, 2018, pp. 369–373.

[125] S. R. Kulkarni, D. V. Kadetotad, S. Yin, J.-s. Seo, and B. Rajendran, "Compact Modelling of Non-Volatile Memory Devices Incorporating Reliability Characteristics," in *SRC TECHCON*, 2019.

[126] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.

[127] S. Raoux, F. Xiong, M. Wuttig, and E. Pop, "Phase change materials and phase change memory," *MRS bulletin*, vol. 39, no. 08, pp. 703–710, 2014.

[128] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[129] Z. Xu, K. B. Sutaria, C. Yang, C. Chakrabarti, and Y. Cao, "Hierarchical modeling of phase change memory for reliable design," in *IEEE 30th International Conference on Computer Design (ICCD)*, Sept 2012, pp. 115–120.

[130] P. Fantini, A. Benvenuti, A. Pirovano, F. Pellizzer, D. Ventrice, and G. Ferrari, "A compact model for phase change memories," in *International Conference on Simulation of Semiconductor Processes and Devices*, Sept. 2006, pp. 162–165.

[131] D. Ventrice, P. Fantini, A. Redaelli, A. Pirovano, A. Benvenuti, and F. Pellizzer, "A Phase Change Memory Compact Model for Multilevel Applications," *IEEE Electron Device Letters*, vol. 28, no. 11, pp. 973–975, Nov 2007.

[132] C.-M. Jung, E.-S. Lee, K.-S. Min, and S.-M. S. Kang, "Compact verilog-A model of phase-change ram transient behaviors for multi-level applications," *Semiconductor Science and Technology*, vol. 26, no. 10, p. 105018, 2011.

[133] L. Xi, S. Zhitang, C. Daolin, C. Xiaogang, and C. Houpeng, "A SPICE model for phase-change memory simulations," *Journal of Semiconductors*, vol. 32, no. 9, p. 094011, 2011.

[134] K. Sonoda, A. Sakai, M. Moniwa, K. Ishikawa, O. Tsuchiya, and Y. Inoue, "A compact model of phase-change memory based on rate equations of crystallization and amorphization," *IEEE Transactions on Electron Devices*, vol. 55, no. 7, pp. 1672–1681, 2008.

[135] R. Warren, J. Reifenberg, and K. Goodson, "Compact thermal model for phase change memory nanodevices," in *11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, May 2008, pp. 1018–1045.

[136] E. Covi, A. Kiouseloglou, A. Cabrini, and G. Torelli, "Compact model for phase change memory cells," in *10th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, June 2014, pp. 1–4.

[137] P. Junsangsri and F. Lombardi, "A new comprehensive model of a phase change memory (pcm) cell," *IEEE Transactions on Nanotechnology*, vol. 13, no. 6, pp. 1213–1225, Nov 2014.

[138] C. Dao-Lin, S. Zhi-Tang, L. Xi, C. Hou-Peng, and C. Xiao-Gang, "A Compact SPICE Model with Verilog-A for Phase Change Memory," *Chinese Physics Letters*, vol. 28, no. 1, p. 018501, 2011.

[139] K. Kwong, L. Li, J. He, and M. Chan, "Verilog-A model for phase change memory simulation," in *9th International Conference on Solid-State and Integrated-Circuit Technology (ICSICT)*, 2008, pp. 492–495.

[140] H. Y. Cheng, M. BrightSky, S. Raoux, C. F. Chen, P. Y. Du, J. Y. Wu, Y. Y. Lin, T. H. Hsu, Y. Zhu, S. Kim, C. M. Lin, A. Ray, H. L. Lung, and C. Lam, "Atomic-level engineering of phase change material for novel fast-switching and high-endurance PCM for storage class memory application," in *IEEE International Electron Devices Meeting*, Dec 2013, pp. 30.6.1–30.6.4.

[141] M. B. Sky, N. Sosa, T. Masuda, W. Kim, S. Kim, A. Ray, R. Bruce, J. Gonsalves, Y. Zhu, K. Suu, and C. Lam, "Crystalline-as-deposited ALD phase change material confined PCM cell for high density storage class memory," in *IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 3.6.1–3.6.4.

[142] X. Yujun, K. Wanki, K. Yerin, K. Sangbum, G. Jemima, B. Matthew, L. Chung, Z. Yu, and C. J. J., "Self-healing of a confined phase change memory device with a metallic surfactant layer," *Advanced Materials*, vol. 30, no. 9, p. 1705587, 2018.

[143] A. Pirovano, A. L. Lacaita, D. Merlani, A. Benvenuti, F. Pellizzer, and R. Bez, "Electronic switching effect in phase-change memory cells," in *Digest. International Electron Devices Meeting,*, Dec 2002, pp. 923–926.

[144] S. Song, Z. Song, C. Peng, L. Gao, Y. Gu, Z. Zhang, Y. Lv, D. Yao, L. Wu, and B. Liu, "Performance improvement of phase-change memory cell using $AlSb_3Te$ and atomic layer deposition TiO2 buffer layer," *Nanoscale Research Letters*, vol. 8, no. 1, p. 77, 2013.

[145] M. Avrami, "Kinetics of Phase Change. I General Theory," *The Journal of Chemical Physics*, vol. 7, no. 12, pp. 1103–1112, 1939.

[146] F. Pellizzer, A. Pirovano, F. Ottogalli, M. Magistretti, M. Scaravaggi, P. Zuliani, M. Tosi, A. Benvenuti, P. Besana, S. Cadeo, T. Marangon, R. Morandi, R. Piva, A. Spandre, R. Zonca, A. Modelli, E. Varesi, T. Lowrey, A. Lacaita, G. Casagrande, P. Cappelletti, and R. Bez, "Novel μtrench phase-change memory cell for embedded and stand-alone non-volatile memory applications," in *Digest of Technical Papers - Symposium on VLSI Technology*, June 2004, pp. 18–19.

[147] R. Annunziata, P. Zuliani, M. Borghi, G. D. Sandre, L. Scotti, C. Prelini, M. Tosi, I. Tortorelli, and F. Pellizzer, "Phase Change Memory technology for embedded non volatile memory applications for 90nm and beyond," in *IEEE International Electron Devices Meeting (IEDM)*, Dec 2009, pp. 1–4.

[148] S. R. Nandakumar, I. Boybat, M. L. Gallo, A. Sebastian, B. Rajendran, and E. Eleftheriou, "Supervised learning in spiking neural networks with MLC PCM synapses," in *75th Annual Device Research Conference (DRC)*, June 2017, pp. 1–2.

[149] X. Fong, Y. Kim, R. Venkatesan, S. H. Choday, A. Raghunathan, and K. Roy, "Spin-transfer torque memories: Devices, circuits, and systems," *Proceedings of the IEEE*, vol. 104, no. 7, pp. 1449–1488, 2016.

[150] A. Driskill-Smith, S. Watts, D. Apalkov, D. Druist, X. Tang, Z. Diao, X. Luo, A. Ong, V. Nikitin, and E. Chen, "Non-volatile spin-transfer torque RAM (STT-RAM): An analysis of chip data, thermal stability and scalability," in *IEEE International Memory Workshop*, 2010, pp. 1–3.

[151] T. Kawahara, K. Ito, R. Takemura, and H. Ohno, "Spin-transfer torque ram technology: Review and prospect," *Microelectronics Reliability*, vol. 52, no. 4, pp. 613–627, 2012.

[152] Y. Zhang, W. Zhao, Y. Lakys, J.-O. Klein, J.-V. Kim, D. Ravelosona, and C. Chappert, "Compact modeling of perpendicular-anisotropy CoFeB/MgO magnetic tunnel junctions," *IEEE Transactions on Electron Devices*, vol. 59, no. 3, pp. 819–826, 2012.

[153] D. C. Ralph and M. D. Stiles, "Spin transfer torques," *Journal of Magnetism and Magnetic Materials*, vol. 320, no. 7, pp. 1190–1216, 2008.

[154] J.-B. Kammerer, M. Madec, and L. Hebrard, "Compact modeling of a magnetic tunnel junction – Part I: Dynamic magnetization model," *IEEE Transactions on Electron Devices*, vol. 57, no. 6, pp. 1408–1415, 2010.

[155] Z. Xu, K. B. Sutaria, C. Yang, C. Chakrabarti, and Y. Cao, "Compact modeling of STT-MTJ for SPICE simulation," in *European Solid-State Device Research Conference (ESSDERC)*, 2013, pp. 338–341.

[156] L.-B. Faber, W. Zhao, J.-O. Klein, T. Devolder, and C. Chappert, "Dynamic compact model of spin-transfer torque based magnetic tunnel junction (MTJ)," in *4th International Conference on Design & Technology of Integrated Systems in Nanoscal Era (DTIS'09)*, 2009, pp. 130–135.

[157] G. D. Panagopoulos, C. Augustine, and K. Roy, "Physics-based SPICE-compatible compact model for simulating hybrid MTJ/CMOS circuits," *IEEE Transactions on Electron Devices*, vol. 60, no. 9, pp. 2808–2814, 2013.

[158] Y. A. Belay, A. Cabrini, and G. Torelli, "A comprehensive Verilog-A behavioral model of Spin-Transfer Torque memory cell," in *Ph.D. Research in Microelectronics and Electronics (PRIME), 2016 12th Conference on*, 2016, pp. 1–4.

[159] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J.-O. Klein, S. Galdin-Retailleau, and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE Transactions on Biomedical C6ircuits and Systems*, vol. 9, no. 2, pp. 166–174, 2015.

[160] A. Vatankhahghadim, S. Huda, and A. Sheikholeslami, "A survey on circuit modeling of spin-transfer-torque magnetic tunnel junctions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 9, pp. 2634–2643, 2014.

[161] J. Kim, A. Chen, B. Behin-Aein, S. Kumar, J.-P. Wang, and C. H. Kim, "A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies," in *IEEE Custom Integrated Circuits Conference (CICC)*, 2015, pp. 1–4.

[162] M. Madec, J.-B. Kammerer, and L. Hébrard, "Compact modeling of a magnetic tunnel junction–part ii: Tunneling current model," *IEEE Transactions on Electron Devices*, vol. 57, no. 6, pp. 1416–1424, 2010.

[163] S. Ament, N. Rangarajana, A. Parthasarathya, and S. Rakhejaa, "Solving the stochastic Landau-Lifshitz-Gilbert-Slonczewski equation for monodomain nanomagnets: A survey and analysis of numerical techniques," *arXiv preprint arXiv:1607.04596*, 2016.

[164] F. Romá, L. F. Cugliandolo, and G. S. Lozano, "Numerical integration of the stochastic Landau-Lifshitz-Gilbert equation in generic time-discretization schemes," *Physical Review E*, vol. 90, no. 2, p. 023203, 2014.

[165] H. Lim, S. Lee, and H. Shin, "A survey on the modeling of magnetic tunnel junctions for circuit simulation," *Active and Passive Electronic Components*, vol. 2016, 2016.

[166] K. Jabeur, F. Bernard-Granger, G. Di Pendina, G. Prenat, and B. Dieny, "Comparison of Verilog-A compact modeling strategies for spintronic devices," *Electronics Letters*, vol. 50, no. 19, pp. 1353–1355, 2014.

[167] K. C. Chun, H. Zhao, J. D. Harms, T.-H. Kim, J.-P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, 2013.

[168] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, p. 165209, 2007.

[169] C. Lin, S. Kang, Y. Wang, K. Lee, X. Zhu, W. Chen, X. Li, W. Hsu, Y. Kao, M. Liu *et al.*, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, 2009, pp. 1–4.

[170] Z. Xu, C. Yang, M. Mao, K. B. Sutaria, C. Chakrabarti, and Y. Cao, "Compact modeling of STT-MTJ devices," *Solid-State Electronics*, vol. 102, pp. 76–81, 2014.

[171] M. Marins de Castro, R. C. Sousa, S. Bandiera, C. Ducruet, A. Chavent, S. Auffret, C. Papusoi, I. L. Prejbeanu, C. Portemont, L. Vila, U. Ebels, B. Rodmacq, and B. Dieny, "Precessional spin-transfer switching in a magnetic tunnel junction with a synthetic antiferromagnetic perpendicular polarizer," *Journal of Applied Physics*, vol. 111, no. 7, p. 07C912, 2012.

[172] O. Kavehei and E. Skafidas, "Highly scalable neuromorphic hardware with 1-bit stochastic nano-synapses," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2014, pp. 1648–1651.

[173] X. Ma, H. Wu, D. Wu, and H. Qian, "A 16 Mb RRAM test chip based on analog power system with tunable write pulses," in *Non-Volatile Memory Technology Symposium (NVMTS)*, 2015, pp. 1–3.

[174] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush, "19.7 A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 338–339.

[175] H. Li, K. S. Li, C. H. Lin, J. L. Hsu, W. C. Chiu, M. C. Chen, T. T. Wu, J. Sohn, S. B. Eryilmaz, J. M. Shieh, W. K. Yeh, and H. S. P. Wong, "Four-layer 3D vertical RRAM integrated with FinFET as a versatile computing unit for brain-inspired cognitive information processing," in *IEEE Symposium on VLSI Technology*, June 2016, pp. 1–2.

[176] S. Lee, J. Song, C. Seong, J. Woo, J.-M. Choi, S.-C. Kwon, H.-J. Kim, H.-S. Kang, S. G. Kim, H. G. Jung, K.-W. Kwon, and H. Hwang, "Full chip integration of 3-D cross-point ReRAM with leakage-compensating write driver and disturbance-aware sense amplifier," in *IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, June 2016, pp. 1–2.

[177] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H. S. P. Wong, "A neuromorphic visual system using RRAM synaptic devices with Sub-pJ energy and tolerance to variability: Experimental characterization and large-scale modeling," in *International Electron Devices Meeting*, Dec 2012, pp. 10.4.1–10.4.4.

[178] D. Garbin, O. Bichler, E. Vianello, Q. Rafhay, C. Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo, "Variability-tolerant Convolutional Neural Network for Pattern Recognition applications based on OxRAM synapses," in *IEEE International Electron Devices Meeting*, Dec 2014, pp. 28.4.1–28.4.4.

[179] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[180] X. Guan, S. Yu, and H.-S. P. Wong, "A SPICE compact model of metal oxide resistive switching memory with variations," *IEEE Electron Device Letters*, vol. 33, no. 10, pp. 1405–1407, 2012.

[181] P.-Y. Chen and S. Yu, "Compact modeling of RRAM devices and its applications in 1T1R and 1S1R array design," *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4022–4028, 2015.

[182] Z. Jiang, S. Yu, Y. Wu, J. H. Engel, X. Guan, and H. S. P. Wong, "Verilog-A compact model for oxide-based resistive random access memory (RRAM)," in *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Sept 2014, pp. 41–44.

[183] H. Li, T. F. Wu, S. Mitra, and H.-S. P. Wong, "Resistive RAM-Centric Computing: Design and Modeling Methodology," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2263–2273, 2017.

[184] G. Gonzalez-Cordero, J. B. Roldan, and F. Jimenez-Molinos, "Simulation of RRAM memory circuits, a verilog-A compact modeling approach," in *Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov 2016, pp. 1–6.

[185] X. Huang, H. Wu, D. C. Sekar, S. N. Nguyen, K. Wang, and H. Qian, "Optimization of TiN/TaOx/HfO2/TiN RRAM arrays for improved switching and data retention," in *IEEE International Memory Workshop (IMW)*, May 2015, pp. 1–4.

[186] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *IEEE International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 16.2.1–16.2.4.

[187] C. C. McAndrew, G. J. Coram, K. K. Gullapalli, J. R. Jones, L. W. Nagel, A. S. Roy, J. Roychowdhury, A. J. Scholten, G. D. Smit, X. Wang *et al.*, "Best practices for compact modeling in Verilog-A," *IEEE Journal of the Electron Devices Society*, vol. 3, no. 5, pp. 383–396, 2015.

[188] A. M. S. Tosson, S. Yu, M. H. Anis, and L. Wei, "A Study of the Effect of RRAM Reliability Soft Errors on the Performance of RRAM-Based Neuromorphic Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3125–3137, Nov 2017.

[189] S. Lashkare, S. Chouhan, T. Chavan, A. Bhat, P. Kumbhare, and U. Ganguly, "PCMO RRAM for Integrate-and-Fire Neuron in Spiking Neural Networks," *IEEE Electron Device Letters*, vol. 39, no. 4, pp. 484–487, April 2018.

[190] S. R. Nandakumar, M. Minvielle, S. Nagar, C. Dubourdieu, and B. Rajendran, "A 250 mV $Cu/SiO_2/W$ Memristor with Half-Integer Quantum Conductance States," *Nano Letters*, vol. 16, no. 3, pp. 1602–1608, March 2016.

[191] P.-Y. Chen *et al.*, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015.

[192] E. C. Ahn, H.-S. P. Wong, and E. Pop, "Carbon nanomaterials for non-volatile memories," *Nature Reviews Materials*, vol. 3, no. 3, p. 18009, 2018.

[193] S. R. Kulkarni, D. V. Kadetotad, S. Yin, J.-s. Seo, and B. Rajendran, "Neuromorphic Hardware Accelerator for SNN Inference based on STT-RAM Crossbar Arrays," to appear in Proceedings of ICECS 2019.

[194] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 1737–1746.

[195] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *Custom Integrated Circuits Conference (CICC)*, sept. 2011, pp. 1–4.

[196] J. Gehlhaar, "Neuromorphic Processing: A New Frontier in Scaling Computer Architecture," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 317–318.

[197] J. S. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Custom Integrated Circuits Conference (CICC)*, Sept 2011, pp. 1–4.

[198] B. Rajendran, Y. Liu, J. Seo, K. Gopalakrishnan, L. Chang, D. Friedman, and M. Ritter, "Specifications of nanoscale devices and circuits for neuromorphic computational systems," *IEEE Transactions on Electron Devices*, pp. 246–253, January 2013.

[199] M. Kraft, A. Kasiński, and F. Ponulak, "Design of the spiking neuron having learning capabilities based on fpga circuits," *IFAC Proceedings Volumes*, vol. 39, no. 17, pp. 301–306, 2006.

[200] G. W. Burr *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017. [Online]. Available: http://dx.doi.org/10.1080/23746149.2016.1259585

[201] D. Kuzum, S. Yu, and P. Wong, "Synaptic electronics: materials, devices and applications," *Nanotechnology*, vol. 24, no. 38, p. 382001, Sept 2013.

[202] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[203] B. L. Jackson, B. Rajendran, G. S. Corrado, M. Breitwisch, G. W. Burr, R. Cheek, K. Gopalakrishnan, S. Raoux, C. T. Rettner, A. Padilla, A. G. Schrott, R. S. Shenoy, B. N. Kurdi, C. H. Lam, and D. S. Modha, "Nano-Scale Electronic Synapses using Phase Change Devices," in *ACM Journal of Emerging Technologies for Computing*, vol. 9, no. 2, Sept. 2013, pp. 12:1–12:20.

[204] G. W. Burr *et al.*, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power)," in *IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 4.4.1–4.4.4.

[205] L. Song *et al.*, "PipeLayer: A pipelined ReRAM-Based Accelerator for Deep Learning," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 541–552.

[206] C. Yakopcic, Z. Alom, and T. Taha, "Memristor crossbar deep network implementation based on a convolutional neural network," in *International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 963–970.

[207] ——, "Extremely parallel memristor crossbar architecture for convolutional neural network implementation," in *International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1696–1703.

[208] M. Suri, O. Bichler, D. Querlioz, O. Cueto, L. Perniola, V. Sousa, D. Vuillaume, C. Gamrat, and B. DeSalvo, "Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction," in *Electron Devices Meeting (IEDM), 2011 IEEE International*, Dec 2011, pp. 4.4.1–4.4.4.

[209] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *arXiv preprint arXiv:1705.08014*, 2017.

[210] D. Garbin *et al.*, "HfO2-Based OxRAM devices as Synapses for Convolutional Neural Networks," *IEEE Transactions on Electron Devices*, vol. 62, no. 8, pp. 2494–2501, Aug 2015.

[211] S. Lim *et al.*, "Adaptive Learning Rule for Hardware-based Deep Neural Networks Using Electronic Synapse Devices," *ArXiv e-prints arXiv:1707.06381v2*, Jul. 2017.

[212] I. Boybat *et al.*, "Neuromorphic computing with multi-memristive synapses," *ArXiv e-prints*, Nov. 2017.

[213] N. Panwar, B. Rajendran, and U. Ganguly, "Arbitrary spike time dependent plasticity (STDP) in memristor by analog waveform engineering," *IEEE Electron Device Letters*, vol. 38, no. 6, pp. 740–743, June 2017.

[214] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[215] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 525–542.

[216] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[217] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[218] O. Valery, P. Liu, and J.-J. Wu, "Low Precision Deep Learning Training on Mobile Heterogeneous Platform," in *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 109–117.

[219] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7675–7684.

[220] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul, "Mixed precision training with 8-bit floating point," *arXiv preprint arXiv:1905.12334*, 2019.

[221] M.-F. Chang, K.-F. Lin, C.-H. Chuang, L.-Y. Huang, T.-F. Chien, S.-S. Sheu, K.-L. Su, H.-Y. Lee, F. T. Chen, C.-H. Lien *et al.*, "Circuit design challenges and trends in read sensing schemes for resistive-type emerging nonvolatile memory," in *IEEE 11th International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2012, pp. 1–4.

[222] C. J. Lin *et al.*, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *2009 IEEE International Electron Devices Meeting (IEDM)*, Dec 2009, pp. 1–4.

[223] Y. Jin, M. Shihab, and M. Jung, "Area, power, and latency considerations of stt-mram to substitute for main memory," in *The Memory Forum, International Symposium on Computer Architecture (ISCA)*, 2014.

[224] H. Cai, W. Kang, Y. Wang, L. Naviner, J. Yang, and W. Zhao, "High performance mram with spin-transfer-torque and voltage-controlled magnetic anisotropy effects," *Applied Sciences*, vol. 7, no. 9, p. 929, 2017.

[225] H. Noguchi *et al.*, "7.2 4Mb STT-MRAM-based cache with memory-access-aware power optimization and write-verify-write / read-modify-write scheme," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 132–133.

[226] S. Kang, W. Y. Cho, B.-H. Cho, K.-J. Lee, C.-S. Lee, H.-R. Oh, B.-G. Choi, Q. Wang, H.-J. Kim, M.-H. Park *et al.*, "A 0.1-μm 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) with 66-MHz Synchronous Burst-Read Operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 210–218, 2007.

[227] Q. K. Trinh, S. Ruocco, and M. Alioto, "Voltage scaled STT-MRAMs towards minimum-energy write access," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 3, pp. 305–318, 2016.

[228] Y. Kim, S. K. Gupta, S. P. Park, G. Panagopoulos, and K. Roy, "Write-optimized reliable design of STT MRAM," in *ACM/IEEE International Symposium on Low Power Electronics and Design*, 2012, pp. 3–8.

[229] A. A. Vyas *et al.*, "International Roadmap of Devices and Systems 2017 Edition: More Moore." *The International Roadmap for Devices and Systems: 2017*, 2018.

[230] S. Moradi and R. Manohar, "The impact of on-chip communication on memory technologies for neuromorphic systems," *Journal of Physics D: Applied Physics*, vol. 52, no. 1, p. 014003, 2018.

[231] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, July 2019.

[232] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[233] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1543–1546.

[234] G. Srinivasan, P. Panda, and K. Rfoy, "SpiLinC: Spiking Liquid-Ensemble Computing for Unsupervised Speech and Image Recognition," *Frontiers in Neuroscience*, vol. 12, p. 524, 2018.

[235] D. Reid, A. J. Hussain, and H. Tawfik, "Financial Time Series Prediction Using Spiking Neural Networks," *PLOS ONE*, vol. 9, no. 8, pp. 1–13, Aug 2014.

[236] N. Soures and D. Kudithipudi, "Deep liquid state machines with neural plasticity for video activity recognition," *Frontiers in Neuroscience*, vol. 13, p. 686, 2019.