University of Tennessee, Knoxville

TRACE: Tennessee Research and Creative Exchange

Doctoral Dissertations

Graduate School

8-2020

Design of Robust Memristor-Based Neuromorphic Circuits and Systems with Online Learning

Sagarvarma Sayyaparaju
University of Tennessee, ssayyapa@vols.utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_graddiss

Recommended Citation

Sayyaparaju, Sagarvarma, "Design of Robust Memristor-Based Neuromorphic Circuits and Systems with Online Learning. " PhD diss., University of Tennessee, 2020.
https://trace.tennessee.edu/utk_graddiss/6821

This Dissertation is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a dissertation written by Sagarvarma Sayyaparaju entitled "Design of Robust Memristor-Based Neuromorphic Circuits and Systems with Online Learning." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

Garrett S. Rose, Major Professor

We have read this dissertation and recommend its acceptance:

James Plank, Nicole McFarlane, Andy Sarles

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Design of Robust Memristor-Based Neuromorphic Circuits and Systems with Online Learning

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Sagarvarma Sayyaparaju

August 2020

© by Sagarvarma Sayyaparaju, 2020
All Rights Reserved.

Dedicated to my parents who have always provided me the necessary strength to move forward in life through their constant support and encouragement.

Acknowledgments

I would like to thank my advisor Dr. Garrett S. Rose for his constant support and cooperation throughout the time I have spent pursuing my PhD at the University of Tennessee. I would also like to thank my doctoral committee members Dr. James Plank, Dr. Nicole McFarlane and Dr. Andy Sarles for serving on my committee and for spending their valuable time. Special thanks to my labmates Md Musabbir Adnan, Sherif Amer, Ryan Weiss, Nicholas Skuda, Samuel Brown, Md Badruddoja Majumder, Mesbah Uddin and Gangotree Chakma for their timely help and interesting discussions throughout my graduate studies. Also, I would like to thank John Reynolds for helping me with some parts of my work.

Abstract

Computing systems that are capable of performing human-like cognitive tasks have been an area of active research in the recent past. However, due to the bottleneck faced by the traditionally adopted von Neumann computing architecture, bio-inspired neural network style computing paradigm has seen a spike in research interest. Physical implementations of this paradigm of computing are known as neuromorphic systems. In the recent years, in the domain of neuromorphic systems, memristor based neuromorphic systems have gained increased attention from the research community due to the advantages offered by memristors such as their nanoscale size, nonvolatile nature and power efficient programming capability. However, these devices also suffer from a variety of non-ideal behaviors such as switching speed and threshold asymmetry, limited resolution and endurance that can have a detrimental impact on the operation of the systems employing these devices. This work aims to develop device-aware circuits that are robust in the face of such non-ideal properties. A bi-memristor synapse is first presented whose spike-timing-dependent plasticity (STDP) behavior can be precisely controlled on-chip and hence is shown to be robust. Later, a mixed-mode neuron is introduced that is amenable for use in conjunction with a range of memristors without needing to custom design it. These circuits are then used together to construct a memristive crossbar based system with supervised STDP learning to perform a pattern recognition application. The learning in the crossbar system is shown to be robust to the device-level issues owing to the robustness of the proposed circuits. Lastly, the proposed circuits are applied to build a liquid state machine based reservoir computing system. The reservoir used here is a spiking recurrent neural network generated using an evolutionary optimization algorithm and the readout layer is built with the crossbar system presented earlier, with STDP based online learning. A generalized framework for the

hardware implementation of this system is proposed and it is shown that this liquid state machine is robust against device-level switching issues that would have otherwise impacted learning in the readout layer. Thereby, it is demonstrated that the proposed circuits along with their learning techniques can be used to build robust memristor-based neuromorphic systems with online learning.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

For many years, researchers have been trying to develop and implement computing systems which could potentially follow the operation of the mammalian brain in terms of data processing with the aim to be able to deploy them in cognitive applications such as pattern recognition and classification [24]. Biological neural systems (particularly the brain) possess the ability to process massive amounts of data at high speeds while consuming very less power/energy. To be able to perform similar tasks using a computer, with the ever increasing amount of data to be handled, traditional computing which is based on von Neumann architecture is facing the wall in terms of data latency and excessive power dissipation, among other things [43]. This is because this architecture of computing stores data separately (from the processing unit) and relies on the efficiency of memory access to speed up operations. This is in contrast to the in-memory style computing model of biological nervous systems wherein the signals are propagated among neurons, which act as the main processing units and the strength of these signals passing between them is determined by the strength of the synapses connecting these neurons. Hence, in the recent times there has been a shift of interest to use this topology/paradigm of computing model especially for cognitive applications (similar to those performed by biological systems). However, in order to implement cognitive applications, software-based approaches for neural network realization are power hungry and demanding in terms of computing resources [76]. Therefore, hardware implemented neural

networks that mimic biological neural systems using neuron circuits to encode information in the form of discrete spikes with synapses forming their interconnections have seen increased interest.

Various circuit techniques have been adopted so far to implement and realize neuromorphic systems (hardware realization of neural networks and systems). These systems comprise of two major processing units: neurons and synapses. Electrical synapses were proposed using resistors [32], capacitors [67], floating gate transistors [85] and static random access memories (SRAMs) [66, 96]. However, these implementations do not have a synapse that is non-volatile and can be programmed in a continuous (analog) fashion efficiently [1]. In 2008, a non-volatile two-terminal resistance switching device was experimentally demonstrated in [103] and this has been called the memristor. The behavior of this device was linked to the theoretical conceptualization of the memristor device in [26]. This device has been shown to be suitable for use as a synapse and also has been shown to be able to implement the bio-inspired local learning rule of spike-timing-dependent plasticity (STDP) [14, 100, 46]. This device's striking features such as its nanoscale size (with the potential to provide bio-plausible dense connectivity [107]), non-volatile nature and its analog programmability [2] with low power consumption has lead to large spread interest in memristive neuromorphic systems.

Recently, a variety of memristor-complementary metal oxide semiconductor (CMOS) based hybrid neuromorphic systems have been developed and even demonstrated experimentally by many groups in the literature for typical neuromorphic applications such as pattern recognition [25, 98, 81, 56, 5, 117]. Many other works have designed and simulated memristive neuromorphic systems with online learning such as [82, 76, 28, 23, 122, 99, 75, 27, 33, 120, 71, 74, 60]. However, many of such implementations have large circuit area overhead to implement synaptic plasticity and to perform online learning. Some of them employ different spike shapes at the input and output of neurons leading to only single step potentiation/depression (instead of the bio-inspired multi-step STDP) and also need extra circuits/silicon area for producing extra spike shapes [82].

Additionally, contemporary memristor devices suffer from a wide range of non-ideal characteristics such as switching speed asymmetry (typically switching in the resistance

decrease direction is much faster compared to that of the resistance increase direction), switching threshold asymmetry (voltage threshold for switching being not equal in either direction), low resolution of resistance states (not being able to achieve sufficient number of stable resistance levels in between both the possible extremes), low endurance (number of switching cycles the device can endure without significant degradation of properties) and the varying device resistance and switching properties across the multitude of available devices in the literature.

All of the above mentioned non-ideal properties can detrimentally affect a memristor-based system and they must be accounted for during circuit design. This work primarily focuses on designing device-aware efficient neuromorphic circuits that are robust when faced with such non-idealities. Designing systems with such circuits can help them cope with such robustness related issues.

1.2 Research Scope and Goal

The work in the domain of memristor based neuromorphic systems can be broadly classified into three categories: devices, circuits and systems. The work and design goals in each category is dependent on the efficiency and limitations imposed by the other categories. These inter-dependencies must be understood and also be carefully considered for efficient design of memristive neuromorphic systems. The scope of this work primarily encompasses the circuits and systems aspect of this domain. The scope of this work is illustrated in Fig. 1.1. Also, while working to develop efficient circuits and systems the goal here is to be wary of challenges imposed by device limitations and their potential impact on circuit and system operation.

The primary goal of this work is to develop device-aware circuits for memristive synapses and neurons. The design process of these circuits must take into account the above mentioned device-level non-ideal behavior that is prevalent in contemporary memristor devices. This implies that these circuits must be robust in the face of such non-ideal or deviant behavior of memristor devices. Also, in doing so, the developed circuits must have efficient circuit level implementation and operation that is on par with the state-of-the-art.

The goal in terms of synapses is to have a synaptic implementation that can act as either excitatory or inhibitory and can undergo online learning without the need for any local control circuitry. This means that no special control blocks must be needed at the synapse level to perform online learning (which would otherwise nullify the density advantage offered by memristors and possibly make a crossbar configuration infeasible). Also, the synaptic online learning must be controllable and robust with respect to device level issues mentioned above.

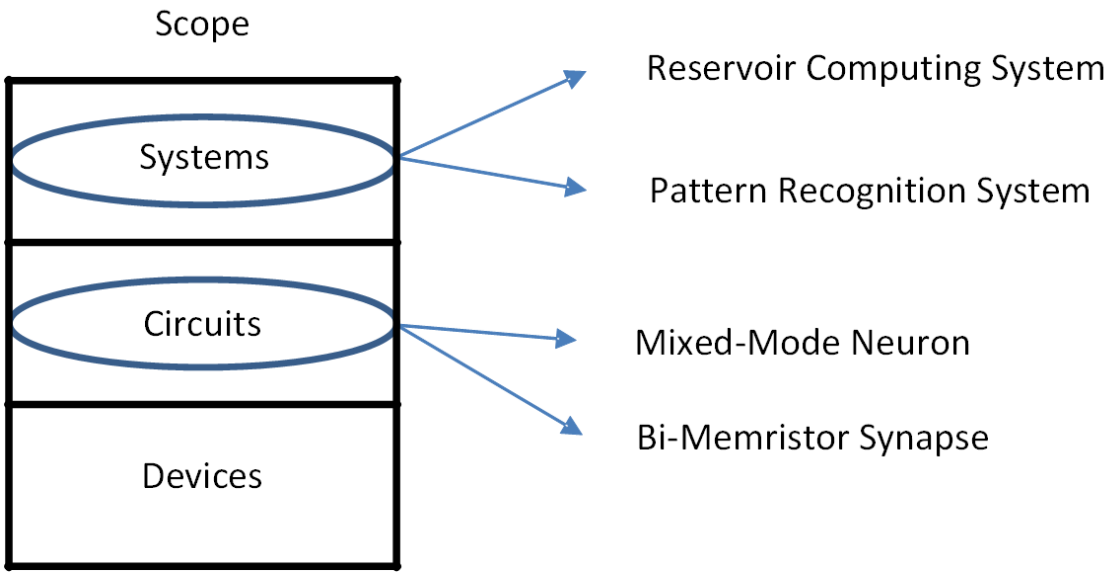

Figure 1.1: Scope of the research work presented here.

For the neuron design, the goal is to make it generic enough such that when the synapse device type is changed, the neuron must not be required to undergo re-design. This implies that the goal is to make the neuron's accumulation rate tunable on-chip so that it may not be needed for it to be custom designed for a given device and it may be tuned to work for a range of devices. This would make the neuron applicable to a plethora of device types currently available.

Finally, it is aimed to demonstrate the robustness of the developed circuits at the system level. Neuromorphic systems such as crossbar based learning systems and even bigger systems such as reservoir computing systems are aimed to be developed using the proposed circuits.

The goal at the system level is to show that these circuits can be utilized to develop efficient memristive neuromorphic systems and that they are robust against prevalent device level issues.

1.3 Research Contribution

The research contributions of this work can be enumerated as below:

- A bi-memristor synapse is designed that is capable of being both excitatory and inhibitory without synapse level control circuits. No additional control blocks are needed to determine the potentiation or depression condition for the synapse. With this synapse, an STDP scheme is designed with spikes shaped such that they are discrete in both time and voltage. The primary advantage of using such spikes is that the designer has a control over both the bias voltage and the time period and they can be tuned on-chip to desired values. This implies that flux input to the synapse during learning may be precisely controlled.

- The proposed synapse's STDP characteristics are shown to be controllable with a clock. By choosing an appropriate clock frequency, it is shown that the magnitude of STDP weight updates may be controlled precisely and fine tuned. Also, it is demonstrated that by modulating the duty cycle of the clock signal used, the device level issues such as switching speed asymmetry may be rectified and mitigated, which would have otherwise crippled the STDP characteristics.

- A mixed-mode neuron is proposed which is generic in terms of the devices with which it can be used. The major advantage offered by this neuron design is that its design and implementation is a one-time process and does not need to be custom designed to suit the specific memristor device under consideration for the given application. It is shown that this neuron's accumulation can be tuned on chip and can be made to work with devices with a variety of resistance values. In this manner the proposed neuron eliminates the need for custom designing and/or re-design of neurons for a

specific memristor and/or application which can prove costly in terms of design and fabrication time, effort and cost.

- A memristive crossbar system with supervised STDP based learning using the proposed circuits is shown. In this crossbar, the bi-memristor synapse is used at each crosspoint. A winner-takes-all (WTA) logic block is also designed for the output neuron stage to determine the 'winning' neuron (based on the highest accumulation among neurons) when an application is run on this system.

- The crossbar system's learning and operation is analyzed for various device level issues such as switching speed and threshold asymmetry, endurance and change of memristor device used in the synapse. It is demonstrated that this system can cope with them and is hence robust against such issues.

- A liquid state machine based reservoir computing system is built using the proposed circuits. This system has the distinction of using evolutionary optimization (EO) generated spiking recurrent neural networks in the reservoir and uses a simple and robust STDP based online learning in the readout layer. The reservoir topology generation process is streamlined by the use of EO algorithm here. Also, the use of STDP learning based memristive crossbar leads to a compact and efficient implementation of the readout layer, which plays a critical role in the reservoir computing system.

- A generic hardware framework for the implementation of the reservoir computing system is proposed. Using this hardware framework, any random topology of the system may be realized.

- The memristive reservoir computing system's learning is analyzed under the presence of device level switching issues in the readout layer. It is shown that by virtue of using the proposed circuits, the system is robust and that it can cope with such issues, which would have otherwise affected it adversely.

1.4 Dissertation Overview

The rest of this dissertation is organized as follows: Chapter 2 gives a brief background of memristor devices and the memristor switching model used here. It also provides a literature review about the prior work on memristive synapses and neurons. Chapter 3 presents the proposed bi-memristor synapse and also analyzes its STDP behavior under various conditions such as clock frequency change and switching speed asymmetry. Chapter 4 presents the proposed on-chip tunable (generic) mixed-mode neuron circuit. Chapter 5 presents a crossbar based system for pattern recognition application using the proposed synapse and neuron. It also demonstrates that the system is robust owing to the use of the proposed circuits. Chapter 6 presents a liquid state machine based reservoir computing system using the proposed synapse and the crossbar system as the readout layer. Lastly, Chapter 7 summarizes all of the work described here and presents the possible directions for related future work.

Chapter 2

Background

2.1 Memristors

In this section, the background of memristor devices is discussed. First, the concept of memristor devices and their operation principle is described and later the switching model used in this work is presented.

2.1.1 Basics of Memristors

In 1971, Leon O. Chua had proposed in his paper [26], that there must be six one-to-one relations amongst the four basic circuit parameters, namely *voltage (V)*, *current (I)*, *charge (Q)* and *flux (ϕ)*. While relationships between voltage & current, charge & voltage and current & flux are defined by resistance, capacitance and inductance respectively, the relation between voltage & flux and current & charge is obtained from their fundamental definitions (charge is the time integral of current while flux linkage is the time integral of voltage). However, there was a missing link between charge and flux. This concept is illustrated in Fig. 2.1. Based on this, Chua postulated that a fourth fundamental passive device (without internal power supplies) must exist. This is considered fundamental because it cannot be constructed by any combination of the other fundamental elements namely resistors, capacitors and inductors. This two terminal device was named a memristor since it acts as a resistor with memory.

In 2008, a team at HP Labs [103] experimentally demonstrated a passive two-terminal (titanium oxide based) resistive switching device that had the properties of a memristor as predicted by Chua. Such memristive devices have since been physically realized and resistive switching has been demonstrated using several types of materials such as oxides [12, 40, 64, 116, 84, 30, 80, 49], chalcogenides [58, 72, 59], silicon-based [18, 50, 57], organic materials [13], ferroelectric materials [21, 73, 47, 15, 41], carbon nanotubes [46, 83, 4], etc. Additionally, volatile memristors have also been discussed [69, 68, 123, 6]. Memristors are particularly attractive for applications such as non-volatile memory arrays and as synapses for neurmorphic systems because of their nanoscale size and their ability to be arranged in a crossbar fashion as shown in Fig. 2.2, thus potentially providing dense connectivity [77] between pre- and post-neurons in a neuromorphic system.

Memristors are essentially resistors whose resistance can be altered by subjecting them to a certain amount of voltage or current flux. This is done by applying a net voltage bias across the device for a certain period of time. When this bias is above a certain threshold, known as the switching threshold voltage (V_{th}), the memristor's resistance changes and the device is said to have switched. The memristor's resistance can have any value between two extremes known as the low resistance state (LRS) and the high resistance state (HRS). These values are dependent on the type of device under consideration (for example, based on the material used and the switching mechanism involved) and its physical dimensions of implementation. The switching of the device in the HRS to LRS direction is known as the positive direction and the switching parameters applicable to this direction appear with a subscript p. Similarly, switching in the LRS to HRS direction is called the negative direction and the parameters applicable here are denoted with a subscript n. It may be noted that memristors are typically insulators in their as-fabricated state and need a one-time forming step to bring them to a conducting/switching state [8]. The physics of switching mechanism in memristors is dependent on the specific material stack (top and bottom electrodes and the switching layer between them) used for the device and and could be unipolar (switching in both directions with same voltage bias polarity) or bipolar (switching in opposite directions needs opposite bias polarity) based on the operating conditions. A review of such mechanisms can be found in [104].

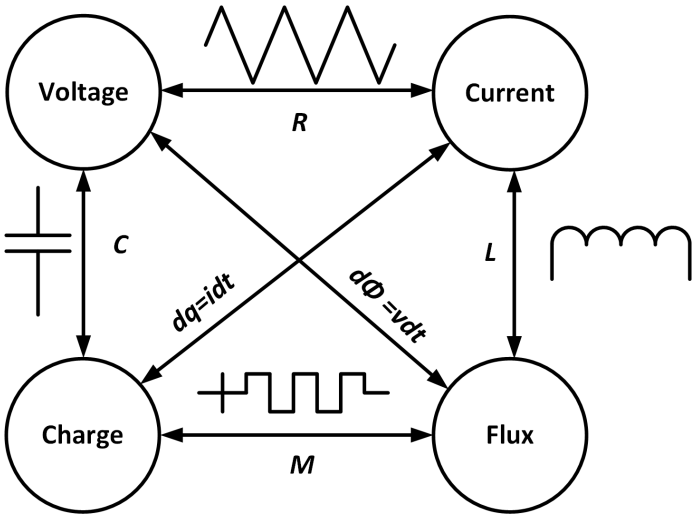

Figure 2.1: The four basic circuit parameters and the relations between them.

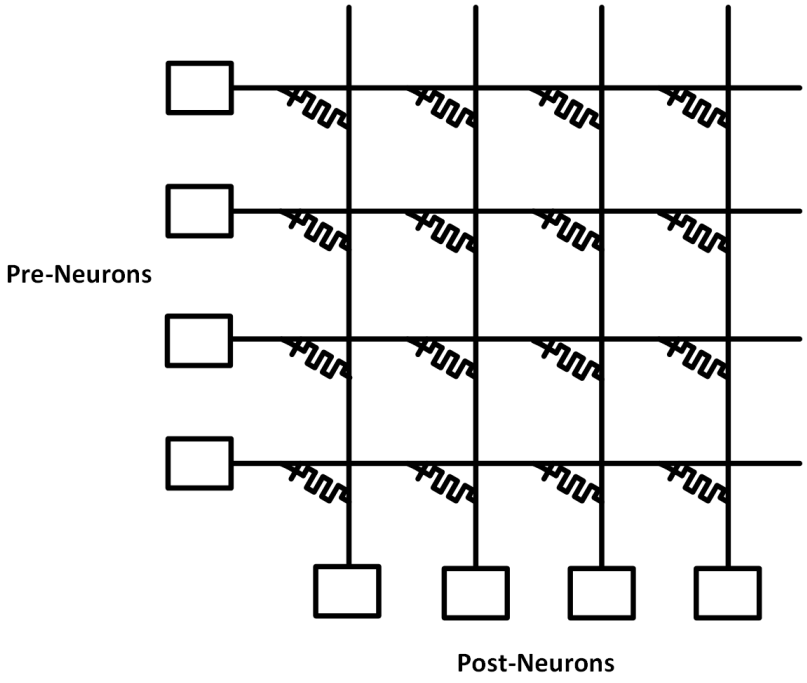

Figure 2.2: An illustration of the (dense) crossbar configuration offered by memristors.

2.1.2 Memristor Switching Model

Various memristor models have been developed ever since they have been discovered, in order to be able to use them for simulation and design purposes. The proposed models thus far are primarily of two types: physics-based and behavioral models. While physics-based models are limited by the scanty development of physical explanation of the switching process in memristors, behavioral models are accurate, amenable to parameter extraction and provide good convergence for simulation purposes. Behavioral models use an internal state variable that depends on applied voltage/current bias. Another relation then links it to the memristor's resistance. However, as the state variable may not be readily measurable, this method may not be parameter extraction friendly.

In [9], a model based on instantaneous resistance as state variable was proposed and this has been used in this work. This is advantageous because resistance may be obtained from the I-V curves of the device characterization data. This model can be mathematically expressed as follows:

$$\frac{dM}{dt} = \begin{cases} -C_{LRS}(\frac{V(t)-V_{tp}}{V_{tp}})^{P_{LRS}} f_{LRS}(M(t)), & V(t) > V_{tp} \\ C_{HRS}(\frac{V(t)-V_{tn}}{V_{tn}})^{P_{HRS}} f_{HRS}(M(t)), & V(t) < V_{tn} \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

where C and p are the speed and non-linearity parameters, respectively. f_{HRS} and f_{LRS} are the window functions to define the resistance change plateauing near to the edges. The equation for the window function is as follows.

$$f(M(t)) = \begin{cases} \frac{1}{1+e^{\frac{M(t)-\theta_{HRS}HRS}{\beta_{HRS}\Delta r}}}, V(t) < V_{tn} \\ \frac{1}{1+e^{\frac{\theta_{LRS}LRS-M(t)}{\beta_{LRS}\Delta r}}}, V(t) > V_{tp} \end{cases} \tag{2.2}$$

Where $\Delta r = HRS - LRS$ and θ & β are fitting parameters of the window function that define the start of the plateauing and the transition slope thereafter, respectively.

2.2 Related Work on Memristive Synapses

Memristive synapses have been proposed in a wide variety of literature. In [46], a memristor based synapse and its weight update scheme based on STDP have been implemented using pulse width modulation, which is exponentially dependent on the time difference between the pre- and post-neuron spikes. A similar technique was proposed in [100] using time division multiplexing (TDM) of pulses. However, these techniques need additional peripheral circuits (local to the synapse) that keep track of the relative timing of the spikes of the pre- and post-neuron. A similar technique based on spike tracking to perform the needed weight update was proposed in [75]. This technique suffers from the same issues as above. In [76], a 1T1R synapse was proposed and an STDP scheme for it was proposed. However, the transistor used as a gating device eliminates the density advantage obtained by employing nanoscale memristors in a given design.

In [51, 27, 33, 120] a single memristor device is used as a synapse and careful shaping of the neuron spike is used to obtain online learning. The pre-neuron spikes in this case are differently shaped as compared to that of the feedback spikes of the post-neuron. However, this scheme would need different circuits to generate different spikes, thus adding to the area overhead of the neurons. In [38], discrete voltage levels based spikes were used. However, this technique could only implement a positive (excitatory) synapse because it used a single memristor and current only flows in a single direction.

In [97], analog spikes were designed and used to perform STDP. Also, the dependence of the STDP characteristics on the spike shape was analyzed. Negative (inhibitory) synapses were also demonstrated by inverting the spike shape. In [54], the neuron was configured as having a positive or negative weight by using a second generation current conveyor circuit. In both of the techniques above, the sign of the synapse weight is based on the neuron, but not intrinsic to the synapse. In [1, 48], a bridge configuration of four memristors was used as the synapse and it was shown that it can have both positive and negative weights. however, this method needed a differential amplifier to infer the synaptic weight information into a proportional current.

For a compact implementation of synapses capable of acting as both excitatory and inhibitory, two-memristor based synapses have been proposed in multiple works. In [98], two memristors driving opposite currents has been proposed. However, the devices here always switched only in a single direction (to tackle the problem of switching asymmetry and abrupt switching in resistance decrease direction), thereby needing a 'sleep cycle' to reset the devices for continual operation at the system level. In [34], a synapse based on two devices was shown, but STDP was not the focus here, and required complex weight update circuitry.

In [19, 3, 20], a twin memristor synapse was proposed that used digital spikes for operation. However, this scheme needed an additional peripheral circuit ('control block') per synapse to determine the relative timing of the pre- and post-neuron before a weight update was performed. A quad memristor based synapse was presented in [90] to tackle asymmetric switching wherein all of the devices switch only in one direction, but it needed bulky control blocks local to the synapse to determine the state of the synapse during each learning cycle. Moreover, the use of a control block local to the synapse eliminates the density advantage obtained with the use of memristors and also renders the crossbar implementation infeasible.

Additionally, the above schemes do not account for the other prevalent issues of today's devices such as asymmetry of switching speed and/or threshold, limited resistance resolution and switching endurance [87]. These issues can hamper the operation of memristor based neuromorphic systems and must be accounted for at the circuit level.

The work presented here proposes a bi-memristor synapse that circumvents the above described issues related to synapse realization. This synapse utilizes two memristors (to be able to realize both positive and negative weights) and has a weight update scheme that does not require any local control circuitry. Additionally, the spike shapes utilized here are on-chip controllable and are shown to implement an STDP that is robust against device level switching issues.

2.3 Related Work on Neuron Design

The neuron is another key component of neuromorphic systems (apart from synapse). Many models for neuron behavior have been proposed, as reviewed comprehensively in [42]. However, most of the spiking neuromorphic systems mentioned therein utilize an analog integrate and fire (IAF) neuron [60]. This neuron operates by integrating the incoming current (from the synapses) and accumulating the resulting charge. The 'accumulated' voltage is compared to a threshold voltage. When this accumulation exceeds the threshold, the neuron 'spikes'. However, with the IAF neuron, the rate of accumulation is a function of the capacitor value used therein. Hence, the neuron must be custom designed for the specific memristor used as the synapse in a given system. Changing the memristor (without changing the rest of the CMOS design) is often desired with neuromemristive systems in order to experiment with several memristor kinds for a given system. In such a case, the IAF neuron must be changed each time, needing a re-design and a re-fabrication of the front-end-of-line (FEOL) as well, which can prove expensive due to the lithographic mask costs involved.

This work proposes a mixed-mode neuron that overcomes this issue of custom designing a neuron for a specific memristor. The proposed neuron's accumulation rate can be tuned on-chip and hence is applicable for use with a variety of memristor devices without having to design for a specific one and without the need for a re-design. This makes it generic in terms of the devices with which it can operate and eliminates the need for re-design when the synaptic device changes.

Chapter 3

Bi-Memristor Synapse

3.1 Synapse Circuit Design

The proposed synapse here [88, 89] consists of two memristors connected between a pre- and a post-neuron as shown in Fig. 3.1(a). In [19] a similar twin-memristor synapse was presented. However, the design there needed control circuitry local to synapse to determine the time difference between pre- and post-neuron spikes for learning. The synapse proposed in this work eliminates the need for such additional circuit overhead. The bi-memristor synapse here operates in two modes: *accumulation* and *learning*. While the post-neuron is accumulating, if the pre-neuron generates a spike, post-neuron's S1 & S2 are closed and it accumulates incoming current as depicted in Fig. 3.1(b). The pre-neuron applies opposite polarity spikes on nodes 1 and 2 while the node on the post-neuron end is biased to a virtual ground. This results in currents that flow in opposite directions in the synapse memristors M_p and M_n, and the resultant current into post-neuron is given by:

$$i = i_{M_p} - i_{M_n} = (G_p - G_n)V_{spike} \tag{3.1}$$

Hence, synaptic weight, given by its effective conductance is:

$$G_{eff} = G_p - G_n = \frac{1}{M_p} - \frac{1}{M_n} \tag{3.2}$$

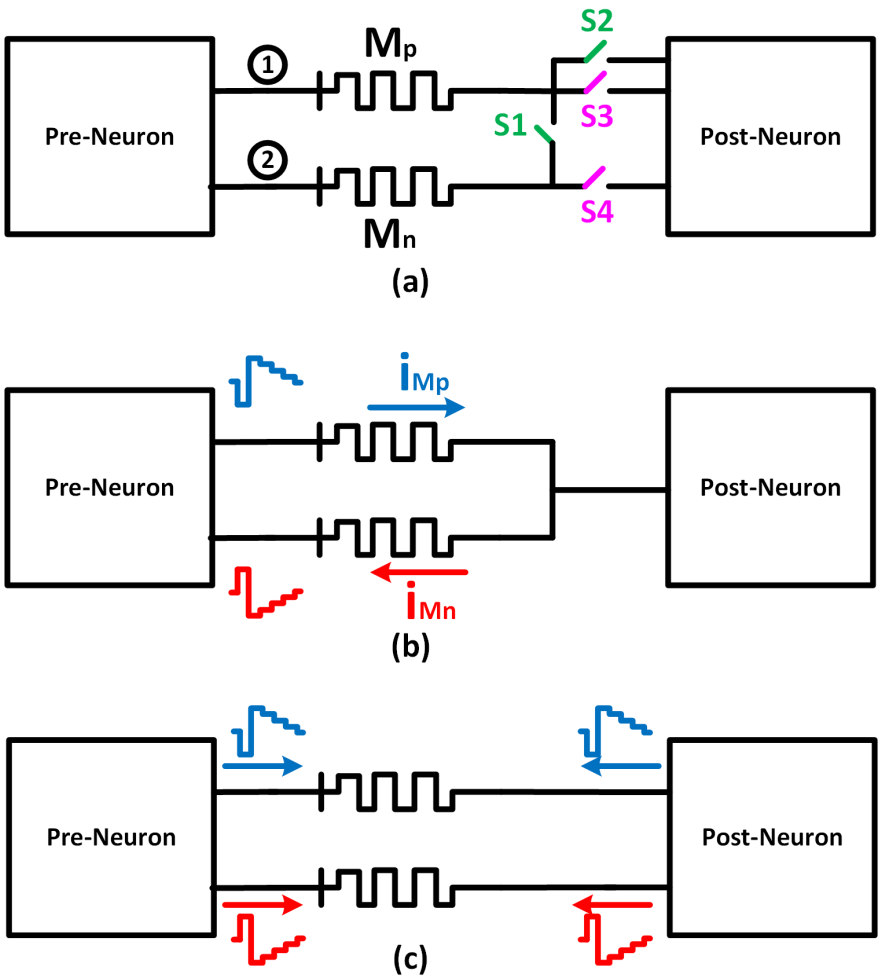

Figure 3.1: (a) The proposed bi-memristor synapse connecting the pre- and post-neuron. (b) The synapse while the pre-neuron is spiking and the post-neuron is in accumulation phase. (c) The synapse when both neurons are in spiking phase.

This current coming into the post-neuron is accumulated therein, and it spikes when the accumulation is greater than threshold. Upon spiking, the post-neuron's S1 & S2 open and S3 & S4 close, giving rise to the topology in Fig. 3.1(c). Also, the neuron not only propagates its spikes to its output synapses, but also sends feedback spikes to its input synapse(s). When this occurs, the synapse enters the *learning* phase, in which both the memristors are biased with spikes on their both ends, leading to a potential weight change depending on the timing of the spikes.

The dependence of the effective voltage bias across the memristors on the relative timing of the neurons' spikes was simulated and is shown in Fig. 3.2. As shown in Fig. 3.2(a), during a potentiation condition, M_p has a resultant positive potential across it, greater than the switching threshold. Similarly, M_n has a net negative voltage across it. Therefore, M_p decreases and M_n increases, resulting in an increase of G_{eff} as shown in (3.2). The new net conductance G'_{eff} can be expressed as follows:

$$\begin{aligned} G'_{eff} &= \frac{1}{M_p - \Delta M} - \frac{1}{M_n + \Delta M} \\ &= \frac{1}{M_p\left(1 - \frac{\Delta M}{M_p}\right)} - \frac{1}{M_n\left(1 + \frac{\Delta M}{M_n}\right)} \\ &= \frac{1}{M_p}\left[1 + \frac{\Delta M}{M_p} + \left(\frac{\Delta M}{M_p}\right)^2 + ...\right] - \frac{1}{M_n}\left[1 - \frac{\Delta M}{M_n} + \left(\frac{\Delta M}{M_n}\right)^2 - ....\right] \\ &= \frac{1}{M_p} - \frac{1}{M_n} + \Delta M\left(\frac{1}{M_p^2} + \frac{1}{M_n^2}\right) + \Delta M^2\left(\frac{1}{M_p^3} - \frac{1}{M_n^3}\right) + ... \\ &= G_{eff} + \Delta M\left(G_p^2 + G_n^2\right) + \Delta M^2\left(G_p^3 - G_n^3\right) + ... \end{aligned} \tag{3.3}$$

Hence, for potentiation, the net conductance increment is given by:

$$\begin{aligned} \Delta G &= G'_{eff} - G_{eff} \\ &= \Delta M\left(G_p^2 + G_n^2\right) + \Delta M^2\left(G_p^3 - G_n^3\right) + ... \end{aligned} \tag{3.4}$$

Similarly, during a depression condition, the post-neuron spike occurs before that of the pre-neuron. In this case, the net bias across M_p and M_n is reversed (compared to potentiation case) shown in Fig. 3.2(b). Hence, in this case M_p increases and M_n decreases resulting in

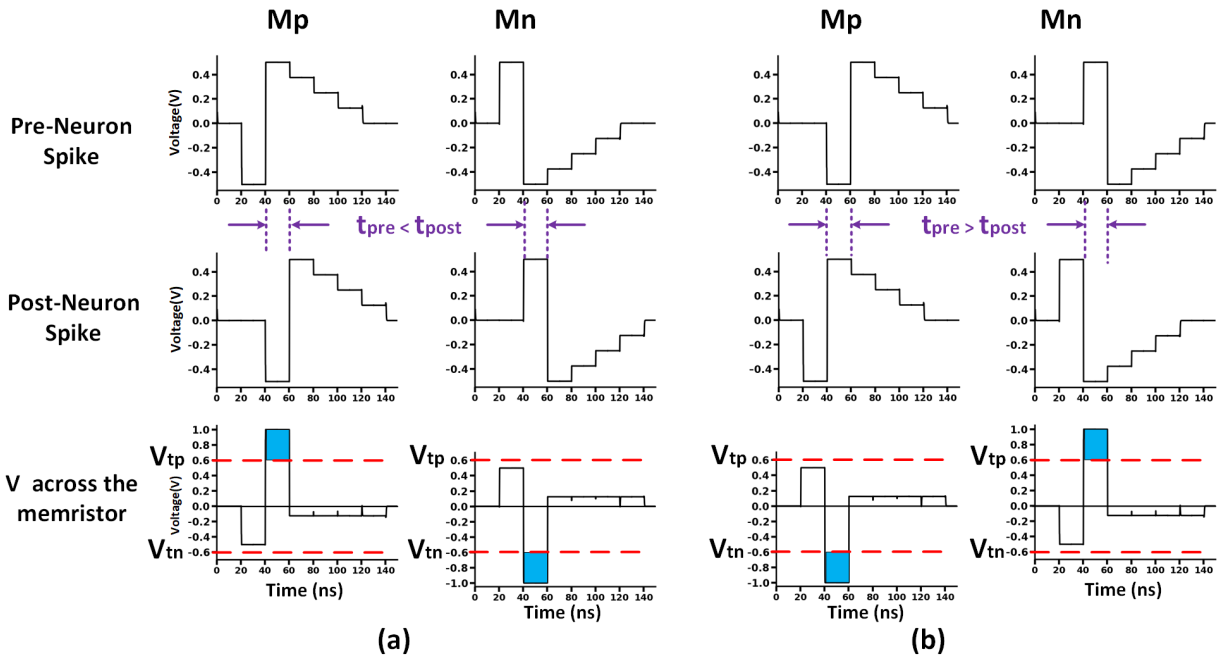

Figure 3.2: The impact of the pre- and post-neuron spikes' temporal relation on the consequent bias across the memristors for (a) potentiation and (b) depression.

an effective decrease of G_{eff}. The conductance decrement here can be expressed as:

$$\Delta G = -[\Delta M(G_p^2 + G_n^2) - \Delta M^2(G_p^3 - G_n^3) + ...] \tag{3.5}$$

The above equations (3.4) and (3.5) for ΔG resemble an exponential function's series expansion, where ΔM is the variable. From Section 2.1.2, it may be seen that the memristance change can be expressed as $\Delta M = kV^p$, where V is the net bias applied on the memristor and k is a switching constant. Therefore, the incremental conductance change may be expressed as:

$$\Delta G = k_1 V^p + k_2 V^{2p} + k_3 V^{3p} + ... \tag{3.6}$$

It can be seen that in the equation (3.6) above, ΔG resembles an exponential dependence on the net bias V. Also, it may be noted that in Fig. 3.2 voltage levels change linearly with time steps. Therefore, a net bias V that changes linearly depending on $\Delta T = t_{post} - t_{pre}$

(the time difference between the pre- and post-neurons' spikes) is applied across the devices. Hence, ΔG has an exponential dependence on ΔT, thus displaying an exponential STDP characteristic.

3.2 Spike Generator Block

Fig. 3.3 depicts the neuron's block diagram. It consists of a spike generator block in addition to the core neuron circuit. When the neuron accumulates, S3 & S4 open while S1 & S2 close, giving rise to a summation of input currents from the synapse. This leads to an effective current inflow into the neuron, leading to an *accumulation*. This accumulation upon crossing the threshold, leads to the neuron spike, during which it is in its *refractory period*. During the refractory period, the spikes are propagated forward to the output synapses and are also fed back to the synapses at its input by virtue of closing S3 & S4 and opening S1 & S2.

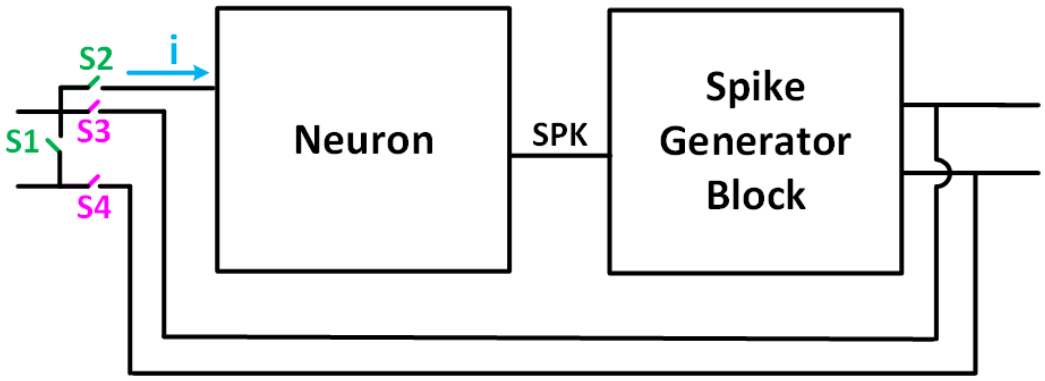

Figure 3.3: Block diagram of the neuron circuit for the proposed synaptic operation.

Fig. 3.4 shows the spike generator block used in the neuron. It shows two switch groups, one to control M_p and one for M_n. The switch controls are taken from the D-flip flop outputs (Q1-Q5). Each one of these outputs controls a switch in the group as shown in Table 3.1.

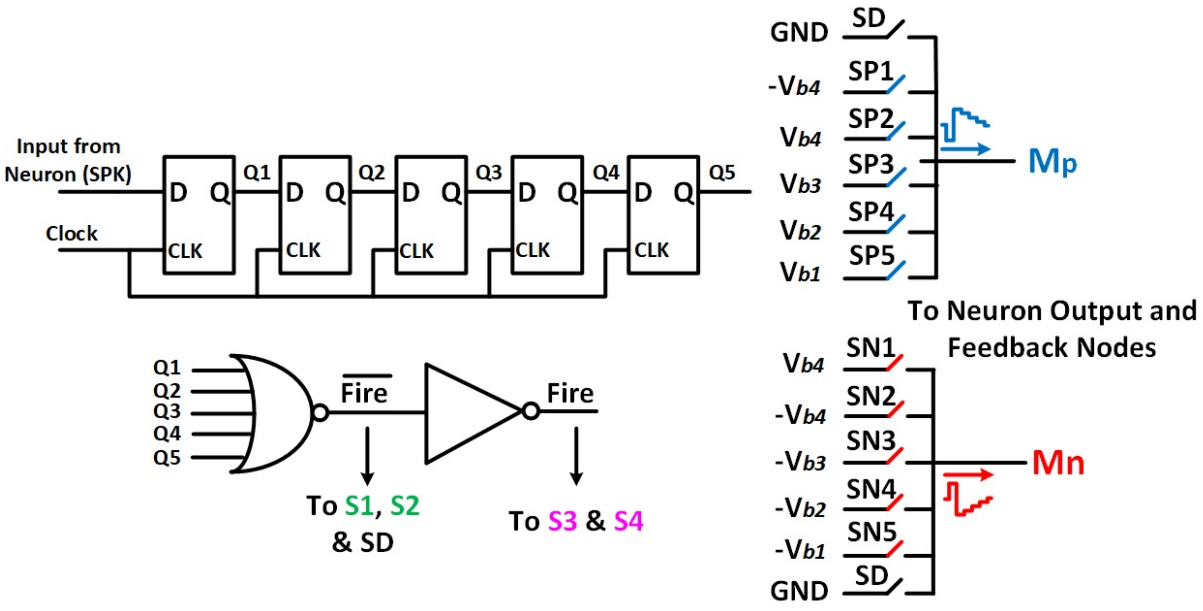

Figure 3.4: Circuit diagram of the spike generator block.

Upon the receipt of a neuron spike, SP1 & SN1 will be first to close. At subsequent clock edges, one of Q1-Q5 will be HIGH, leading to the closure of one of the switches in each group (while the others will remain open). During the time when a switch is closed, the output is biased with the voltage supplied at that switch. These supply voltages may be produced on-chip or may come from off-chip. When there is no spike, the switch SD will be closed, thereby applying a mid-rail GND bias to help mitigate sneak currents issue.

Table 3.1: The dependence of the switches' activity on the flip-flop outputs and the resultant output voltage.

Flip-Flop output 'HIGH'	Switch Closed	Output Voltage
Q1	SP1, SN1	-Vb4, Vb4
Q2	SP2, SN2	Vb4, -Vb4
Q3	SP3, SN3	Vb3, -Vb3
Q4	SP4, SN4	Vb2, -Vb2
Q5	SP5, SN5	Vb1, -Vb1
None	SD	GND

3.3 STDP Behavior

3.3.1 Characteristic STDP Behavior

In order to study the characteristic STDP property of the synapse, simulations have been performed in Cadence Virtuoso using Spectre as the simulator in it. The CMOS portions were implemented using a 65nm design kit and a Verilog-A model for the memristor model in Section 2.1.2 was developed. For this simulation, the following memristor device parameters have been used: $HRS = 12K\Omega, LRS = 2.5K\Omega, V_{tp} = 0.6V, V_{tn} = -0.6V, \theta_{HRS} = 0.85, \theta_{LRS} = 1.6, \beta_{HRS} = 0.07, \beta_{LRS} = 0.07, C_{HRS} = 9.5 \times 10^9, C_{LRS} = 9.5 \times 10^9, P_{HRS} = 2, P_{LRS} = 2$. These values are based on a fit of the proposed memristor model to the device presented in [105] wherein the device was experimentally shown to be switching with intermediate states.

The setup in Fig. 3.5 has been used to simulate the STDP behavior, consisting of synapses S1 and S2 interconnecting the neurons N1-N3 and N2-N3, respectively. The synapse S2 is set to its highest weight state so that N2's spike event leads to a large enough accumulation in N3 to make it spike as well. Thus N3 spike timing is fixed and that of N1 is varied to obtain STDP in S1 synapse. The initial weight of S1 is set to zero, by making M_p=M_n. The results of the STDP performed by using a clock frequency of 50MHz is shown in Fig. 3.6.

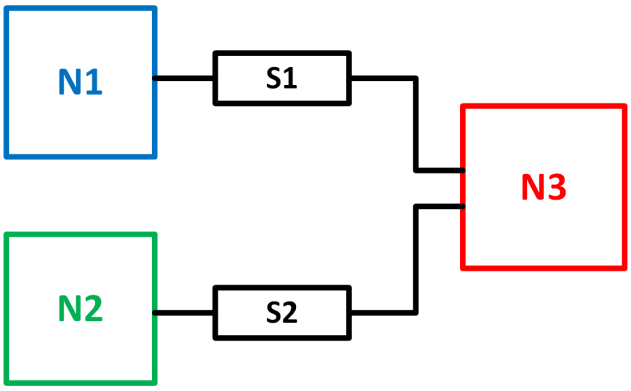

Figure 3.5: The circuit topology used for simulating STDP character of the synapse.

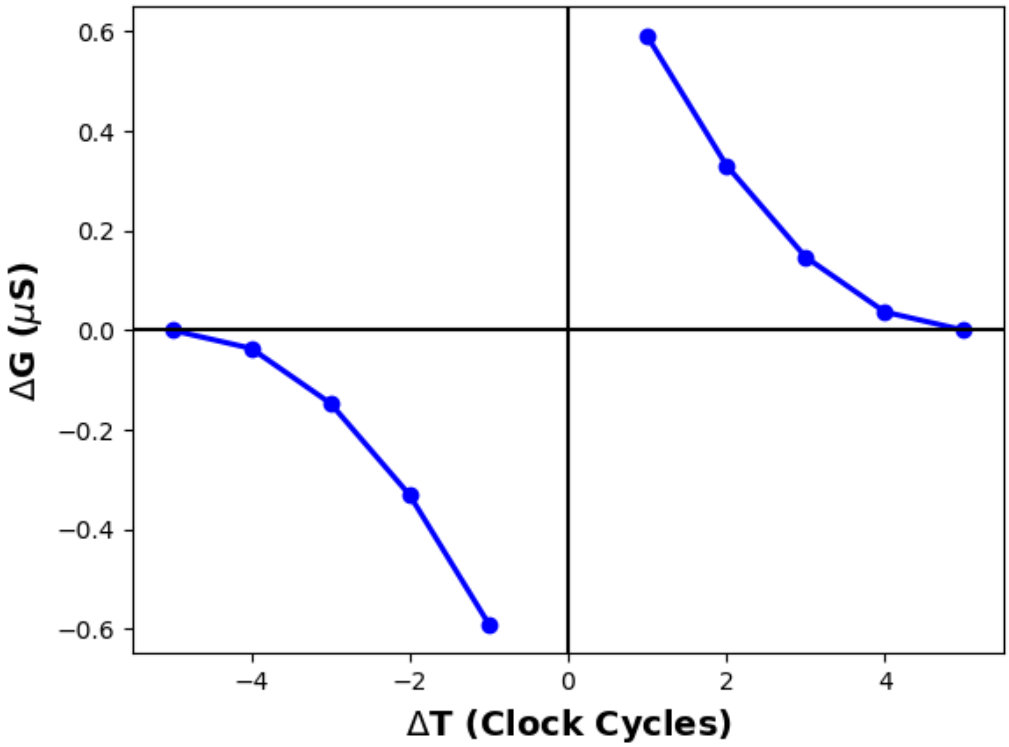

Figure 3.6: The simulated STDP property of the proposed synapse.

3.3.2 Impact of Clock Frequency

As shown in Fig. 3.2, the neuron's spike length in time is dependent on the clock frequency used. The higher the clock period, the longer time the voltage bias is applied across the synapse during learning, leading to a higher ΔM during learning. Therefore, by choosing an appropriate value of the frequency of the clock in the system we can change/control the STDP amount in the synapse. Fig. 3.7 shows the simulated behavior for the dependence of STDP on the clock frequency used. It can be observed that a low clock frequency (high clock period) results in higher ΔG and a steeper STDP curve, whereas the reverse happens for higher clock frequency. Also, at high frequencies, where ΔM becomes small, the magnitude of the higher order terms in Equation (3.4) become negligible, hence reducing it to: $\Delta G \approx (G_p^2 + G_n^2)\Delta M = k\Delta M$, indicating a linear STDP character. This trend is also evident in Fig. 3.7 at higher clock frequencies.

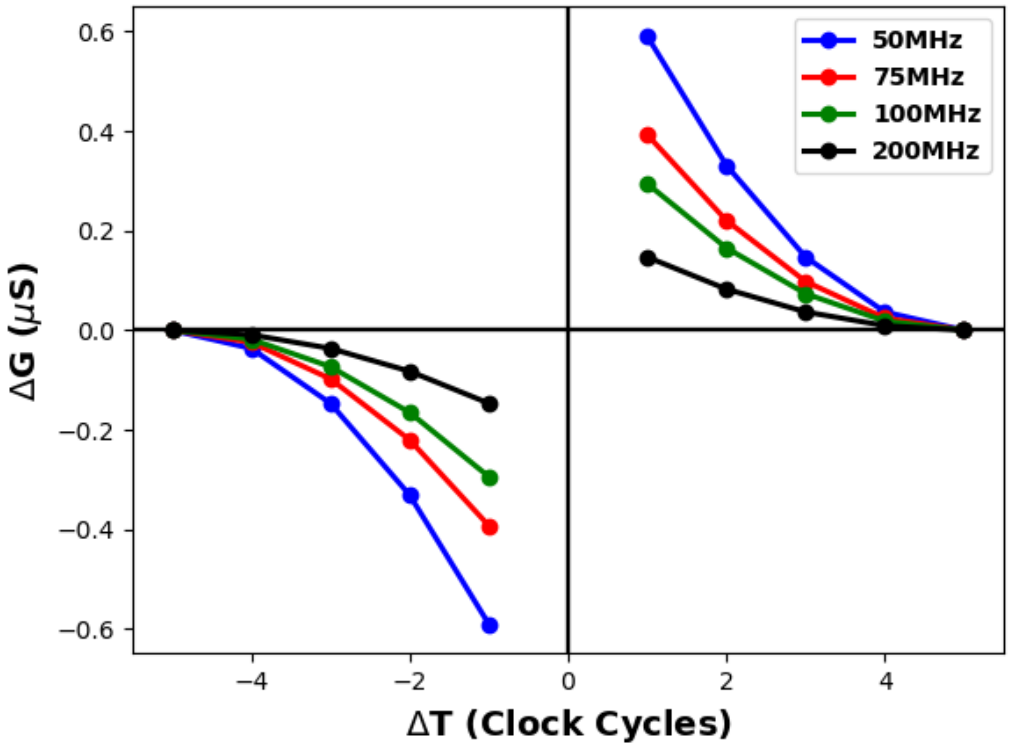

Figure 3.7: Dependence of STDP characteristics of the proposed synapse on the clock frequency used.

3.3.3 Impact of Device Switching Rate Asymmetry

The resistance switching mechanism in memristors is fundamentally different in the resistance increase direction and in the decrease direction. For many devices, this results in the switching occurring at a faster rate in one of the directions. This difference in switching speeds in either direction can be upto two orders of magnitude [35, 11] and this property can prove detrimental for learning purposes since it uses both directions of switching. When this asymmetry is large, the faster switching direction will see much larger steps of resistance change and will tend to dominate the synaptic weight change. These high increments in synaptic weight cripple the STDP characteristics as shown in Fig. 3.8 and also affect the resolution of ΔG during learning, thus having a detrimental impact on the final weights learnt by a system.

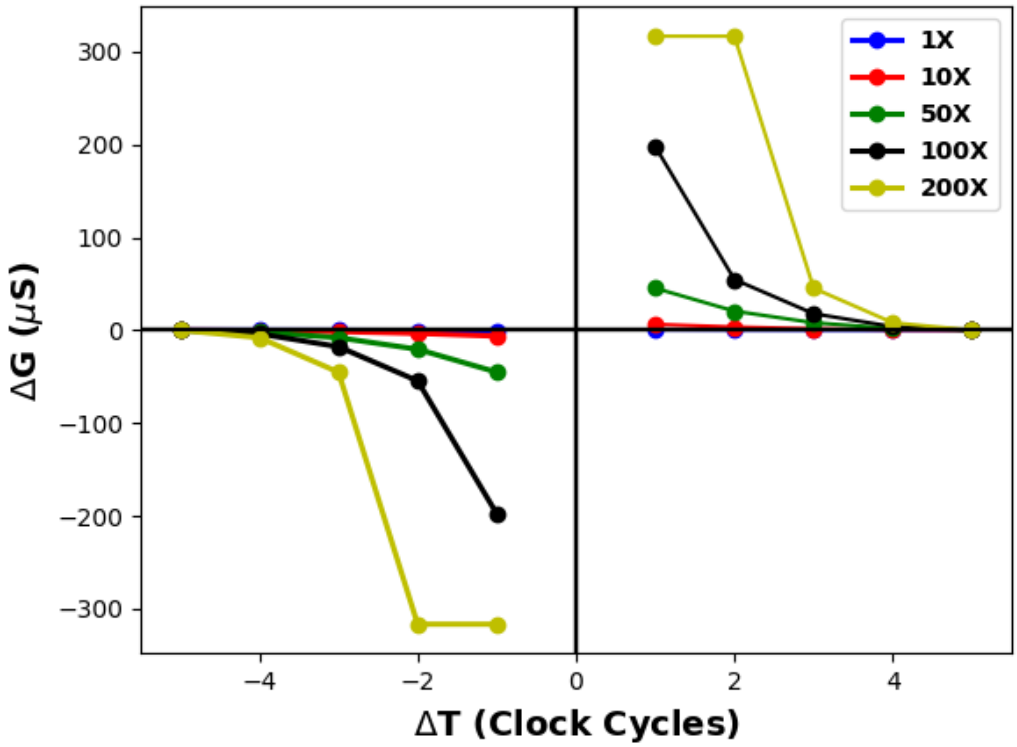

Figure 3.8: Impact of the device's switching speed asymmetry on STDP. Here X = C_{LRS}/C_{HRS}.

The detrimental impact of switching asymmetry can be alleviated by modifying the voltage flux applied in the direction of fast switching. The discretized spike shape utilized here lends itself to this kind of flux modulation applied to the memristor devices. This is done by duty cycle modulation of the spike shape. Fig. 3.10 shows the simulations for the case when decrease of resistance is faster compared to resistance increase ($C_{LRS} > C_{HRS}$). In this case, the neuron's output spikes (propagated to the onward synapses) that are responsible for accumulation are left unaltered, whereas only the feedback spikes (sent to the input synapses) are modulated. Fig. 3.10 shows the spikes shapes after modification and Fig. 3.9 shows the rectified STDP behavior of the synapse after the use of this duty modulation. The reasoning for the shaping of these spikes is as follows: during learning, memristance decrease occurs when a positive bias is applied across it. Such a bias requires a positive bias from the pre-neuron end and a negative bias from the post-neuron end. Therefore, the negative

voltage part of the feedback spikes from the post-neuron is modulated in time to provide less flux as shown in Fig. 3.10. To generate this modulation, S3 and S4 are controlled to be open when the bias is to be shortened in time. To achieve this curtailing in time, its control signal ('*Fire*' signal in Fig. 3.4) is logically ANDed with a duty cycle modulated clock. A modulation in the duty cycle can be achieved from a given system clock using a delay circuit as in [63]. Using this circuit, a original clock can be delayed (say 'd-CLK') and then logical operation (d-CLK⊕CLK)·CLK gives us the desired duty modulated clock. Therefore, by virtue of using the spikes shapes employed here, one can mitigate the effects of switching rate asymmetry.

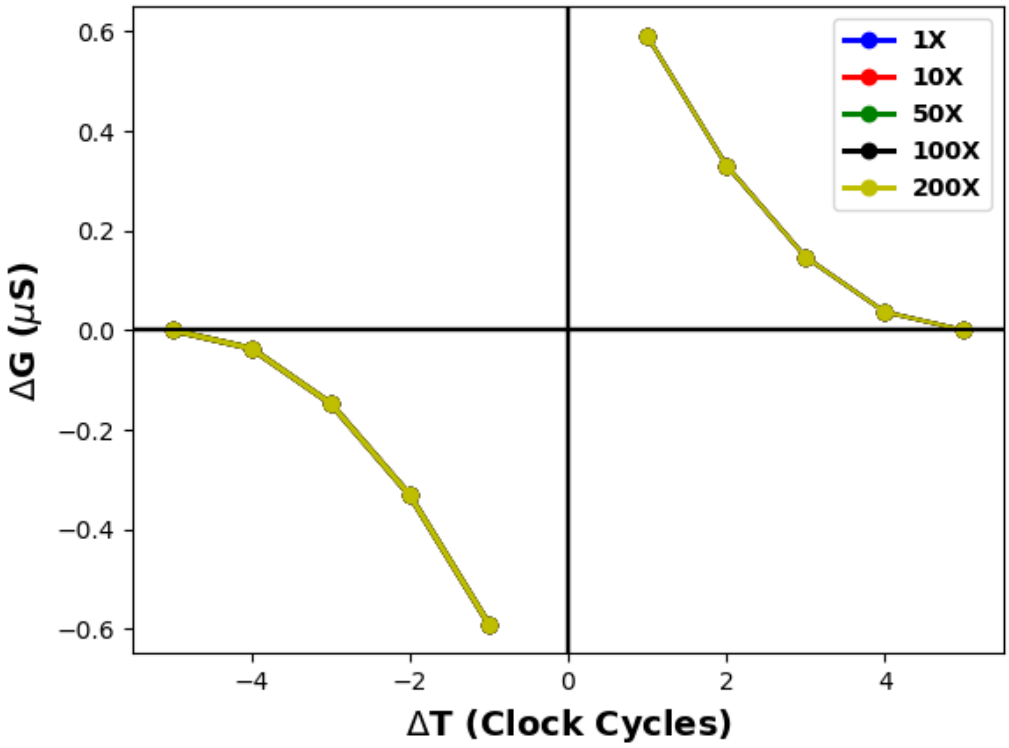

Figure 3.9: STDP reverting to normalcy after duty modulated pulses are used.

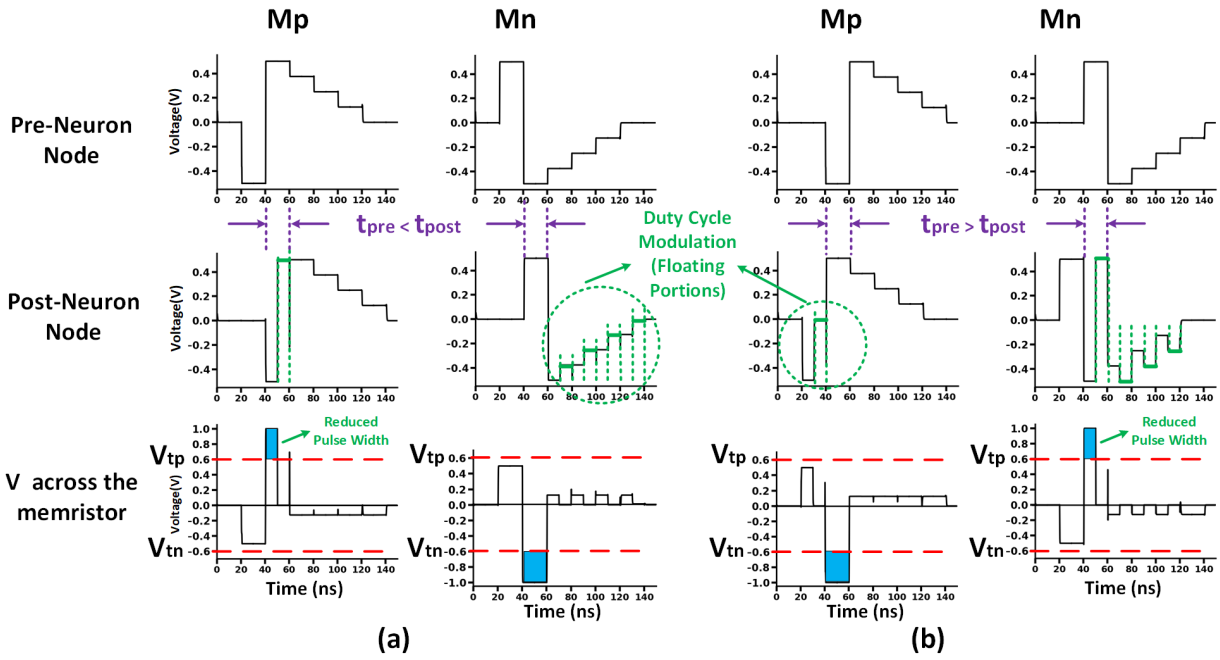

Figure 3.10: Use of duty modulated pulses in the feedback spikes of post-neuron to mititgate switching speed asymmetry impact for (a) Potentiation (b) Depression.

Chapter 4

Mixed-Mode Neuron

In this chapter, the design of an on-chip tunable mixed-mode neuron is presented. First, the functioning of the conventional and widely used integrate and fire neuron is analyzed in the context of applying it for use with multiple types of memristors. Later, the proposed neuron design is presented and its operation is explained.

4.1 Problem with Integrate and Fire Neuron

Traditional IAF neurons perform 'accumulation' from the input current by using a capacitive integrator and store the resulting voltage as V_{mem} as shown in Fig. 4.1. The accumulated voltage can be written as $V_{mem} = -\frac{1}{R_{in}C_{int}} \int_0^t V_{in} dt$. The magnitude of this V_{mem} is dependent on the C_{int}; with higher capacitance resulting in lower magnitude of V_{mem}. Therefore, this indicates that C_{int} value needs to be calculated and customized for the synapse (R_{in}) being used. Currently a wide variety of memristor devices exist and are being used for system level applications. This means that for every new memristor device type, the neuron and/or the system needs to be custom designed. To study this further, three devices are considered here with widely varying resistance values as shown in Table 4.1. These devices have all been experimentally shown to have switching characteristics with stable intermediate states between LRS and HRS.

Fig. 4.2 shows the neuron's response when it was designed to work with Mem_2. The input synapse is set to have five discrete weights (set as G_{max}/x, where x=1-5 and G_{max} =

Table 4.1: The memristance values used here for neuron tunability study.

	Mem_1[105]	Mem_2[61]	Mem_3[44]
LRS	2.5kΩ	15kΩ	250kΩ
HRS	12kΩ	150kΩ	2.5MΩ

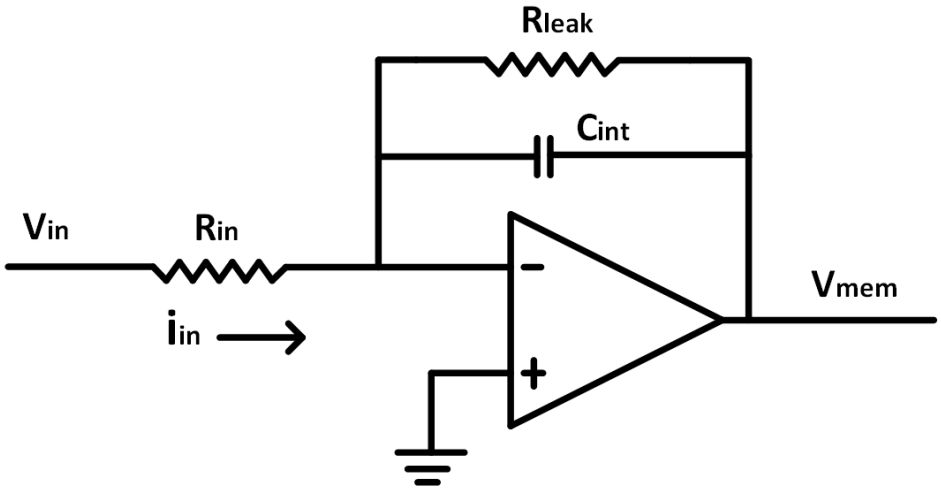

Figure 4.1: A typical integrate and fire (IAF) neuron's integrator circuit.

$\frac{1}{LRS} - \frac{1}{HRS}$) and it can be seen here that the accumulation rate is proportional to the synapse weight; with higher weights (darker shades in Fig. 4.2) resulting in faster accumulation. The neuron designed here was used with the other memristor types. When Mem_1 is used, since its resistance is below that of Mem_2, it inputs more current. For this high current, the capacitor designed for Mem_2 proves too low for Mem_1, leading to high accumulation rates and subsequently high V_{mem} to work with. On the contrary, when Mem_3 was used here, because Mem_3 has higher resistance than Mem_2, the input current to the neuron was very low. The designed capacitor for Mem_2 turned out to be too high for for this device, thereby resulting in too low accumulated voltages to detect and decipher.

The above analysis indicates that this neuron needs to be custom designed to work for a specific memristor device. Owing to the large variety of contemporary memristor device types that exist in literature, and each of them having its own advantages, the design and development of systems that utilize these devices consists of custom design and re-fabrication

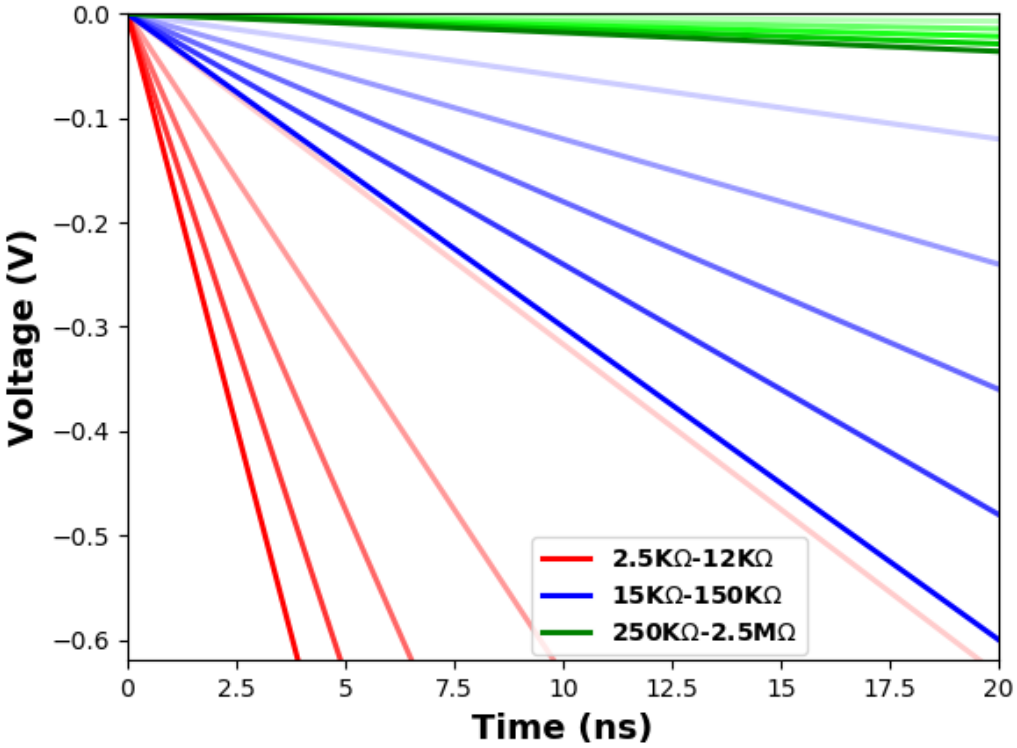

Figure 4.2: The 'accumulated' V_{mem} in the IAF neuron for various device types. For each device type, various synpase weights were simulated. Here, lighter color shade implies a lower synaptic weight and a darker shade is for a higher weight.

of a system for new memristors, which can prove expensive in terms of the time to design, the needed designer effort and the fabrication expenses (for the lithographic masks). To circumvent such a limitation, a mixed-mode neuron with on-chip tunability is proposed here that can adapt to a variety of devices.

4.2 Proposed Mixed-Mode Neuron

The block diagram of the proposed mixed-mode neuron is shown in Fig. 4.3. The incoming synaptic current is converted into an n-bit digital value by the analog block (n=3 here), proportionate to its magnitude. The 'digital block' accumulates and holds this digital value.

This value is then compared with an input digital threshold value and a spike is generated when the threshold is exceeded by the stored digital value.

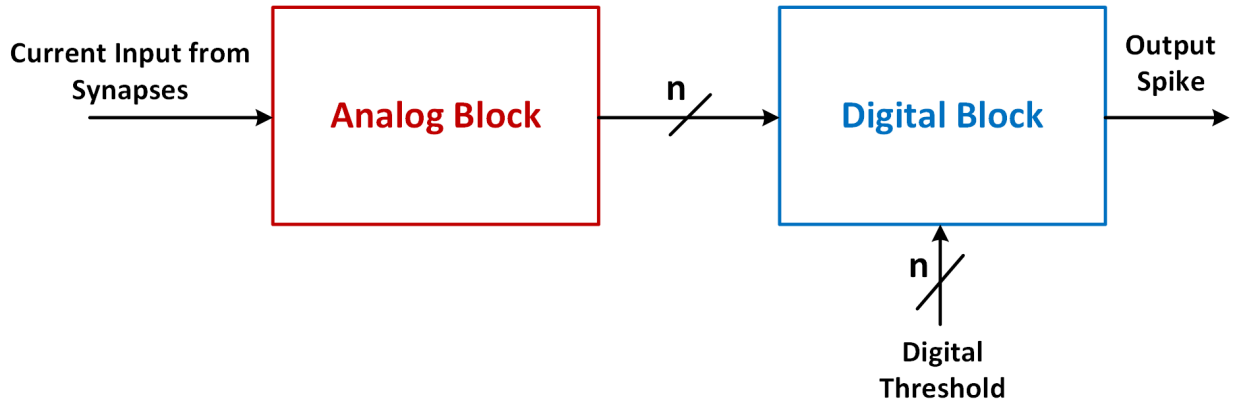

Figure 4.3: The proposed mixed-mode neuron's two constituent blocks.

4.2.1 Analog Block

The details of the analog block of the mixed-mode neuron are depicted in Fig. 4.4. It contains a transimpedance amplifier, to convert the input synaptic current i_{in} into a corresponding voltage V_{mem} (it may be noted here that V_{mem} is negative because the operational amplifier is employed in an inverting configuration). V_{mem} is provided to a 'dynamic CMOS block' that encodes this voltage based on comparison with some input reference voltages. For high input current magnitude, V_{mem} is high in magnitude and hence a high digital value is assigned. It must be noted that since V_{mem} value generated here is a function of the tunable resistor R_{tune}, it may be realized using tunable MOS devices (in linear region) such that R_{tune} may be tuned to provide consistent V_{mem} for a given abstract synaptic weight independent of the synapse's implementation. Hence, with on-chip tunability this mixed-mode neuron can work with a wide variety of devices and have the same accumulation for all of them, without the need for a re-design.

The dynamic CMOS block (Fig. 4.5) comprises of transistor pairs acting as dynamic circuits, each of which is responsible for determining if the input current is within a certain range. Each pair of transistors consist of a 'pre-charge' and a 'discharge' transistor, which

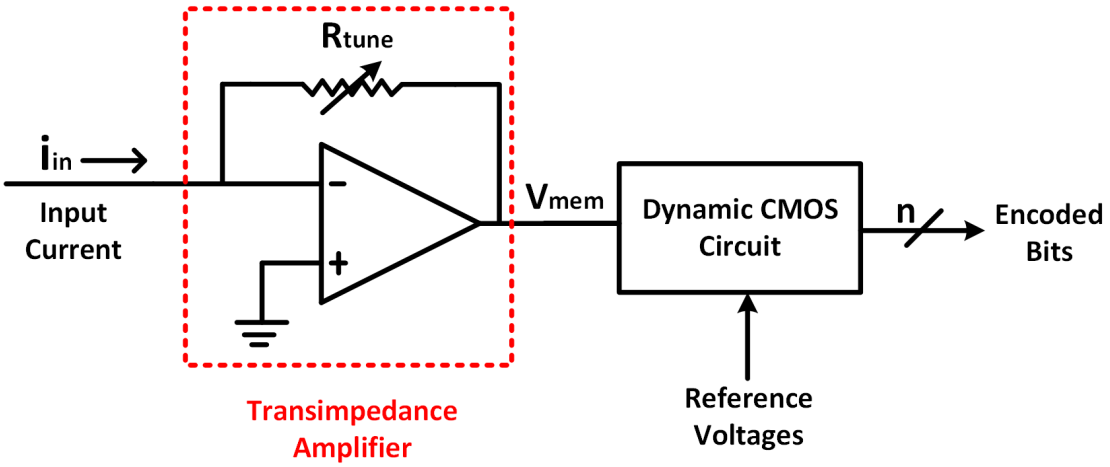

Figure 4.4: Circuit design of the analog block used for digital encoding.

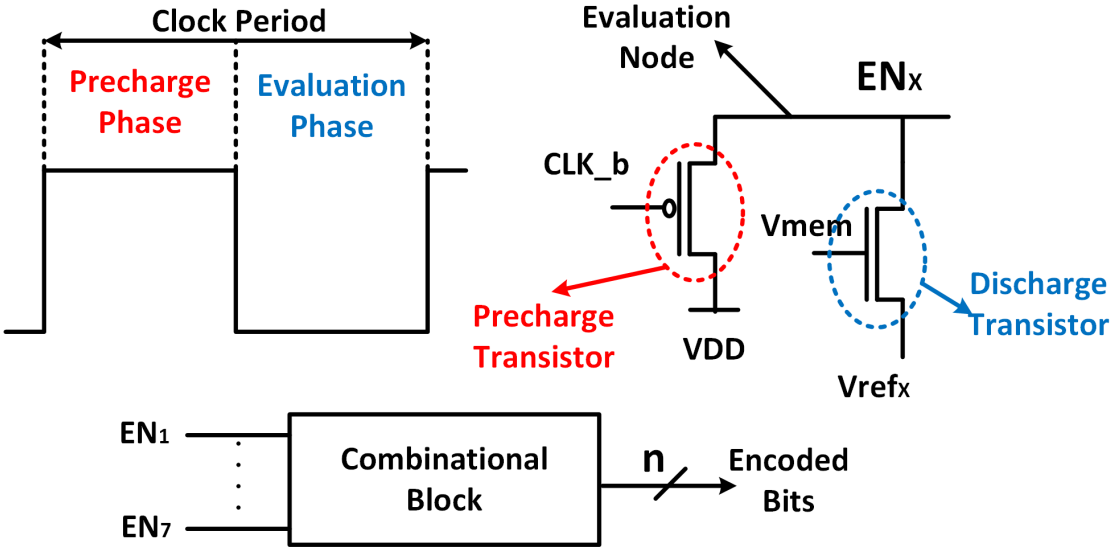

Figure 4.5: The dynamic CMOS based circuit for digital encoding of the V_{mem}. Here, x=1-7. The EN_x nodes' states are used for encoding as delineated in Table 4.2.

perform the function of charging and discharging the *evaluation node*, respectively. During the first half of a clock period, the pre-charge transistor charges up the evaluation node. During the second half of the clock period, this transistor is switched OFF. The discharge transistor takes-in V_{mem} as its gate voltage and is provided with a (input) reference voltage at its source terminal. Thus, V_{mem} primarily determines if this transistor switches ON or OFF. When V_{mem} is high enough in magnitude, it switches ON this transistor and discharges that node. Various V_{ref} values are graded such that $V_{ref1} < V_{ref2} < ... < V_{ref7}$. Thus, for various synaptic currents, various V_{mem} values are generated. The higher the current, the more negative the V_{mem} gets, the tougher it is for the transistor to turn ON. Therefore, the number of fully discharged nodes will be less by the time the evaluation phase ends. This indicates that for a high input current, less evaluation nodes discharge and for low currents, more nodes discharge. At the end of the evaluation phase, the combinational logic block takes-in the status information from the nodes and encodes into a digital value as shown in Table 4.2. Fig. 4.6 presents the simulation results for the evaluation nodes for the case of a '100' as the encoded digital value. It may be seen here that only EN_{1-3} are fully discharged. Also, Fig. 4.7 shows that when the memristor type is changed in the synapse, R_{tune} is tuned on-chip such that the nodes display the same discharge characteristics. This result illustrates that the mixed-mode neuron can be tuned on-chip to produce the same behavior for various memristor devices without the need for re-design.

Table 4.2: The digital encoded values versus the number of discharged nodes.

Nodes Discharged	Encoded Value
EN_{1-7}	000
EN_{1-6}	001
EN_{1-5}	010
EN_{1-4}	011
EN_{1-3}	100
EN_{1-2}	101
EN_1	110
None	111

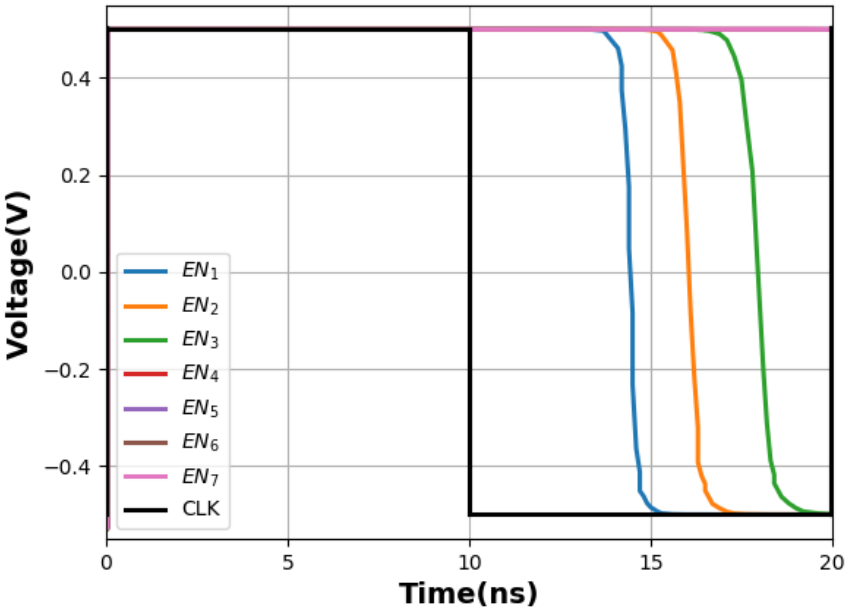

Figure 4.6: Simulation waveforms for the dynamic nodes for the case of a '100' encoded value. Note here that EN_{1-3} are completely discharged.

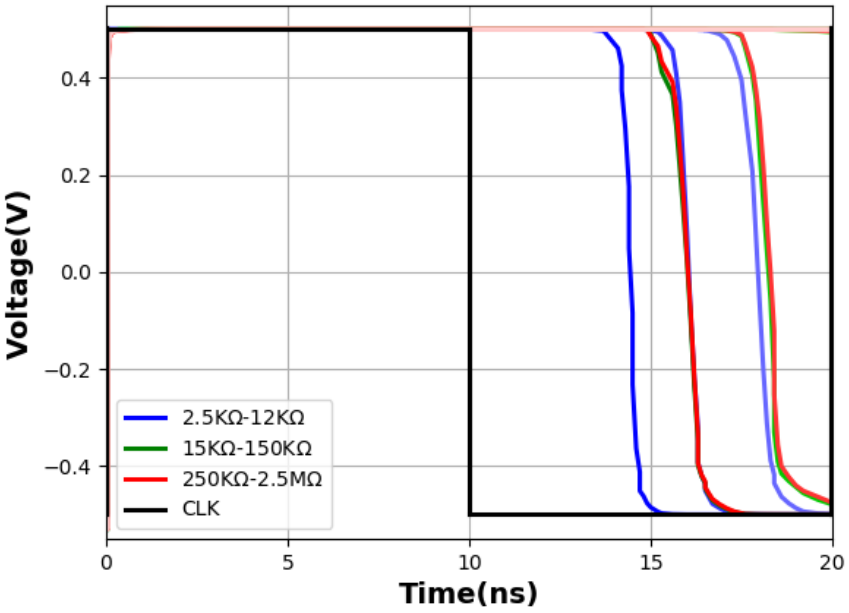

Figure 4.7: Waveforms of the nodes for the three memristor types in the synapse with the tuning of the neuron's accumulation.

4.2.2 Digital Block

The encoded values from the analog block are input to a 'digital block' wherein the 'accumulate and compare' operations are performed in the digital mode. A block diagram of this digital portion of the neuron is shown in Fig. 4.8. The digital accumulation here is compared to an input threshold (also a digital value) and a spike is produced based on the comparison result. Also, when the digital adder has an overflow condition, a spike is produced. The comparator here determines both the comparison result with the input threshold and also keeps a check on the overflow condition of the adder, and decides if a spike must be generated. When the comparator indicates that a spike must be produced, the registers are RESET for the whole length of the refractory period of the neuron.

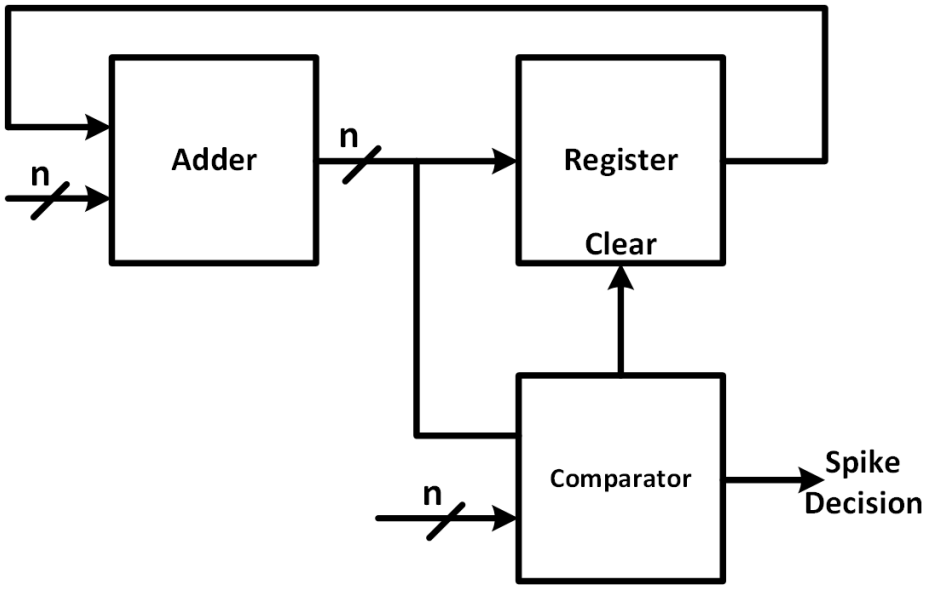

Figure 4.8: Block diagram for the digital portion of the proposed neuron.

4.3 Neuron Test Structures

A test structure is a standalone circuit entity designed with the aim of performing various characterization tests on it. It is usually designed as a single circuit (without any peripherals as much as possible) and connected to input/output (i/o) probe pads directly. Test signals are directly provided to the circuit to perform its characterization using the i/o pads. For the neuron design presented above, four test structures having circuits [91] bearing close resemblance to it were designed and fabricated in CMOS 65nm technology. The details of each of these test structures are presented as follows. A more detailed explanation of the physical layout of these test structures (including their pin out) along with their testing strategies and test results can be found in Appendix A.

4.3.1 Analog Block Test Structure

A test structure to test the analog block by itself (without the digital block cascading it) was developed. This analog block's circuit design is conceptually similar to that described in Section 4.2.1 and is same as that presented in [91]. The schematic of the test structure is shown in Fig. 4.9. As seen in this figure, this test structure comprises of the analog block of the neuron driven by a resistive synapse. A resistive synapse has been used here to have a completely CMOS based design. The synaptic weight here is set to a positive value and since it is a resistor based synapse, its weight is fixed here (determined by the resistors' values). This positive weight synapse drives positive current into the analog block. The block then encodes this current based on the values of the voltage references used (which influence the dynamic CMOS evaluation nodes inside it). Testing can be performed on this test structure by changing the reference voltages to change the discharging pattern among the evaluation nodes, thereby resulting in different encoded values. Fig. 4.10 shows the layout for this test structure.

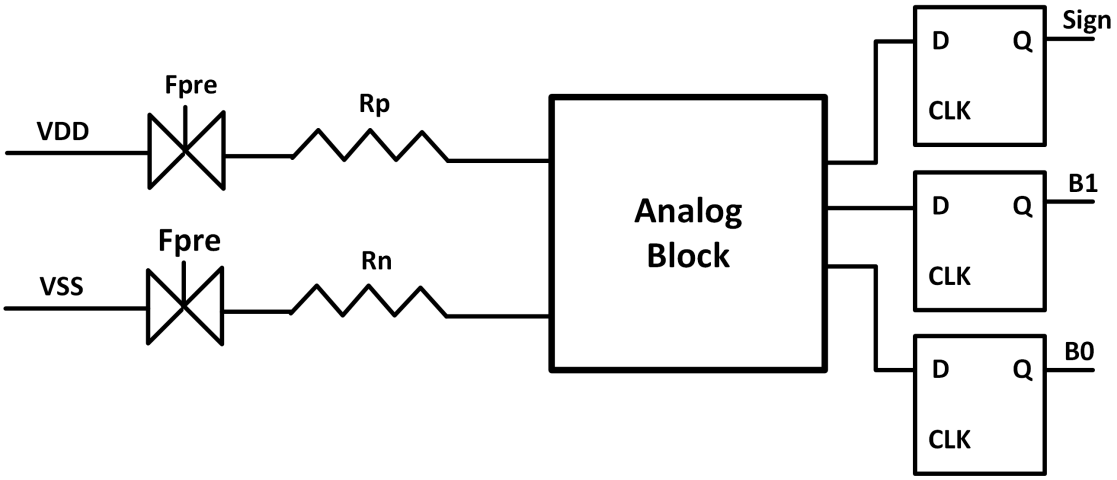

Figure 4.9: The block diagram of the analog block test structure.

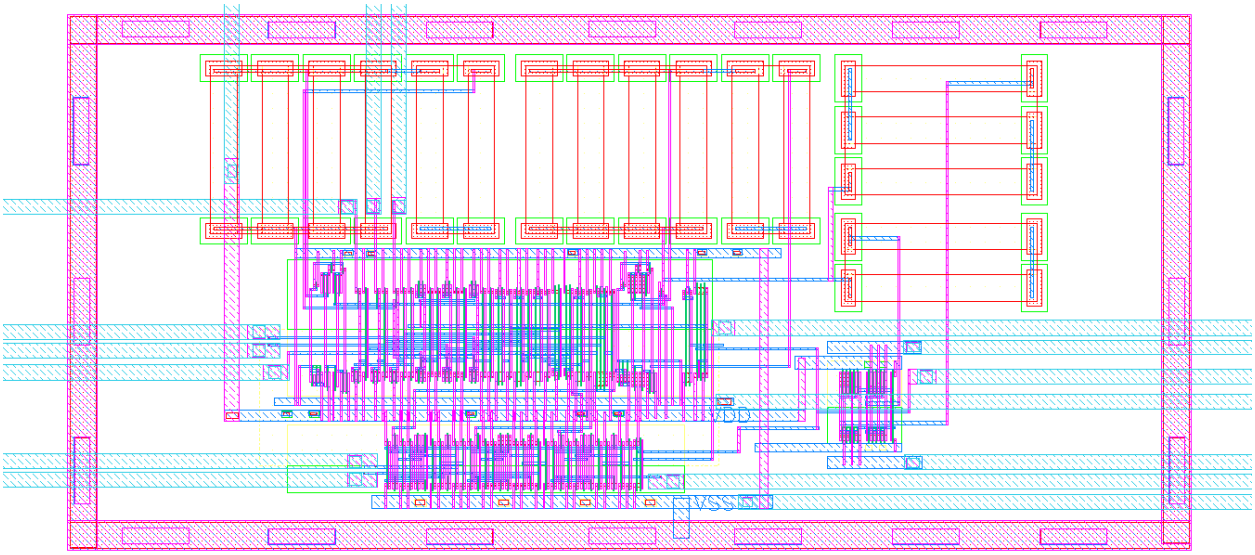

Figure 4.10: The layout of the analog block test structure.

4.3.2 Mixed-Mode Neuron Test Structure with Single Resistive Synapse

In this test structure the analog block is combined with the digital block to form the complete mixed-mode neuron. This test structure's block diagram is shown in Fig. 4.11. Here, as can be seen, the neuron has one resistive synapse (with positive weight) at the input. This input synapse will drive positive current into the neuron. Based on the reference voltages applied, the digital encoding varies. This leads to digital accumulation in the digital block. Based on the digital threshold applied, the neuron will produce a spike. It will produce this spike more frequently for a lower threshold and vice versa. Also, for a given threshold, the spiking frequency is dependent on the encoding performed by the analog block inside it. For higher encoding, the spike is expected to occur more often and vice versa. The layout for this test structure is shown in Fig. 4.12. Note that in Fig. 4.11, the output 'Fre' is the spike output of the neuron and 'Fire' is one clock cycle delayed version of this spike.

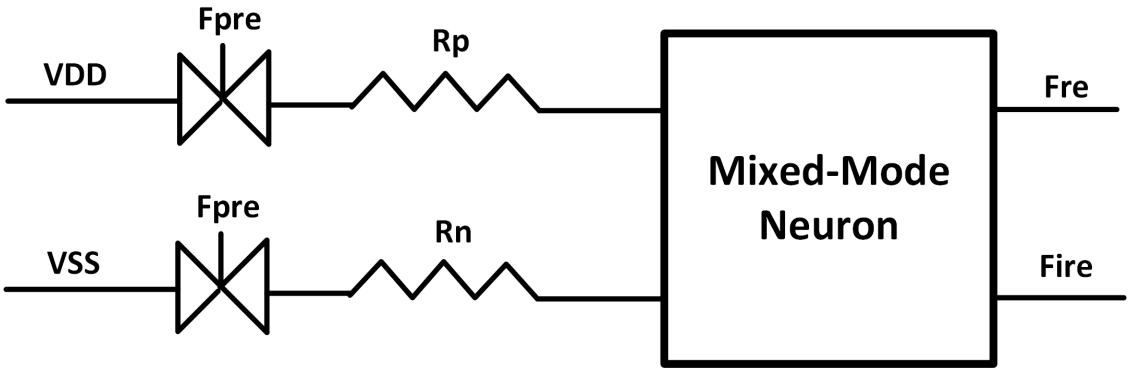

Figure 4.11: The block diagram of the mixed-mode neuron test structure with one resistive synapse.

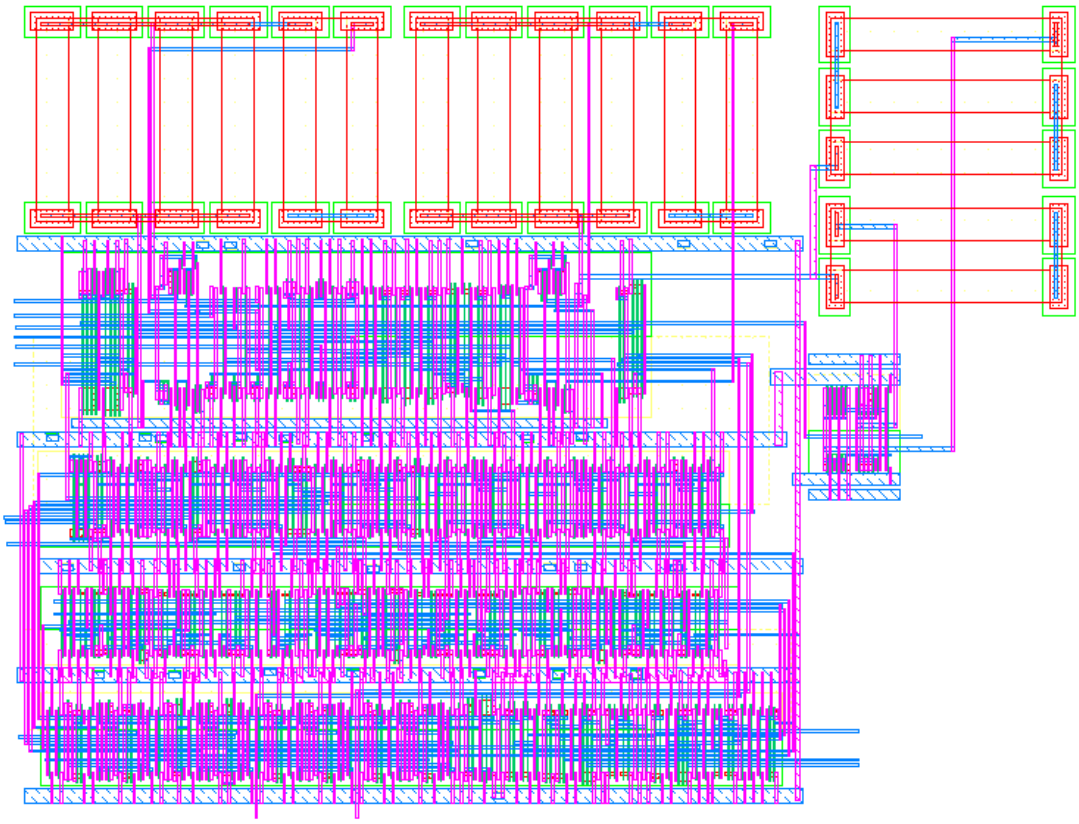

Figure 4.12: The layout of the mixed-mode neuron test structure with one resistive synapse.

4.3.3 Mixed-Mode Neuron Test Structure with two Resistive Synapses

In this test structure (shown in Fig. 4.13), the mixed-mode neuron has two resistive synapses at its input. The purpose of having this test structure is to test the operation of the neuron with multiple input synapses. Note that in this case since resistive synapses have been used, their weight is fixed. Each synapse here has the same positive weight as was in the previous test structure. Since the enable signal ('Fpre') for each synapse is different, each of them may be independently enabled and tested with the neuron. The layout for this test structure is shown in Fig. 4.14.

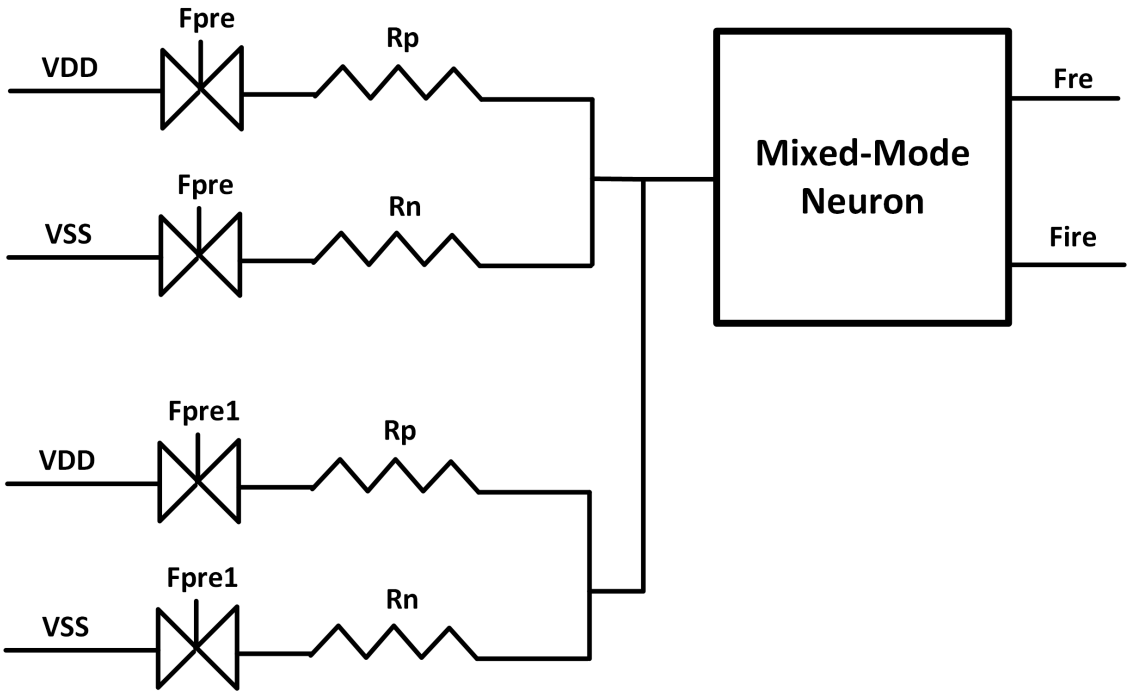

Figure 4.13: The block diagram of the mixed-mode neuron test structure with two resistive synapses.

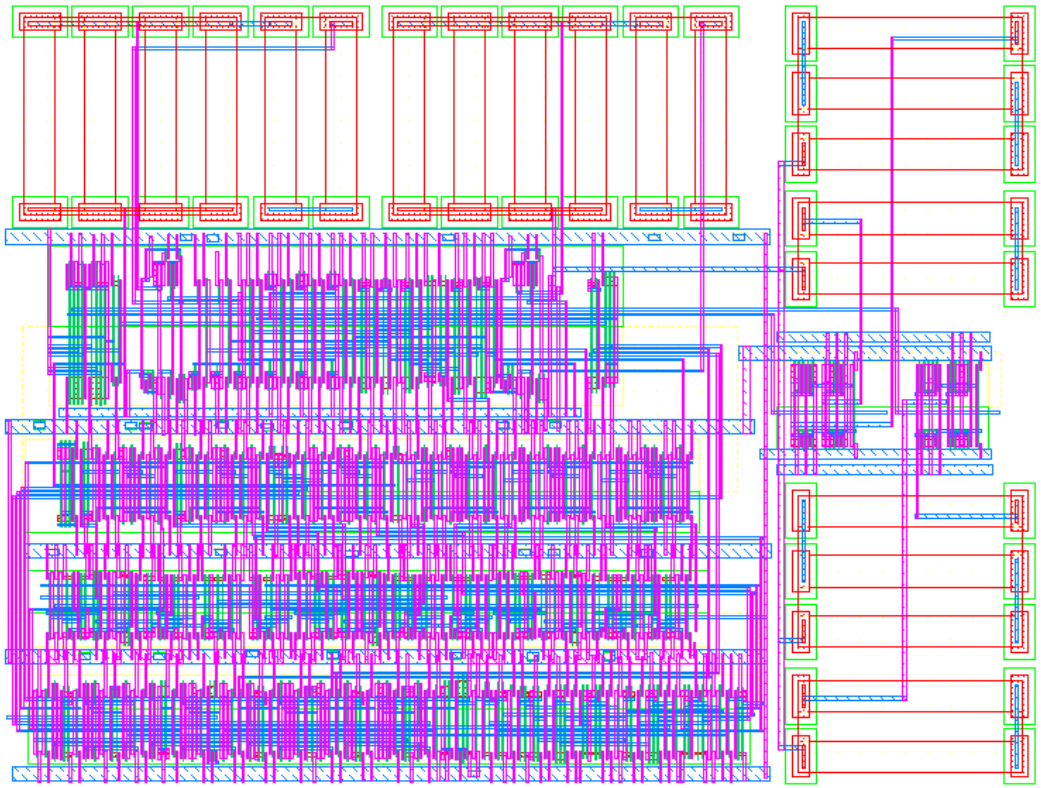

Figure 4.14: The layout of the mixed-mode neuron test structure with two resistive synapses.

4.3.4 Mixed-Mode Neuron Test Structure with a Memristive Synapse

In this test structure, as shown in Fig. 4.15, the neuron has a memristor based synapse at the input. It consists of two memristors (instead of resistors as in the above test structures) along with their forming/programming circuits. These circuits are necessary to carry out an initial forming step [7] on the memristors, which brings them from the as-fabricated insulating state to the LRS state. They may then be programmed to the desired state by the same forming/programming circuit. In this case the memristors may be programmed such that M_p is in LRS and M_n is in HRS, to give a positive weight. The accumulation in the digital neuron can then be tested as was done for the case with resistive synapses. The layout for this test structure is shown in Fig. 4.16.

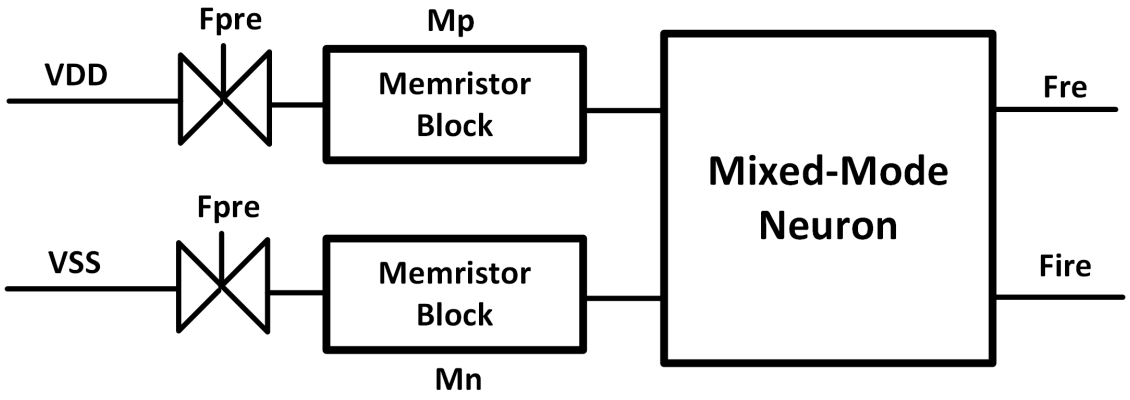

Figure 4.15: The block diagram of the mixed-mode neuron test structure with memristive synapse.

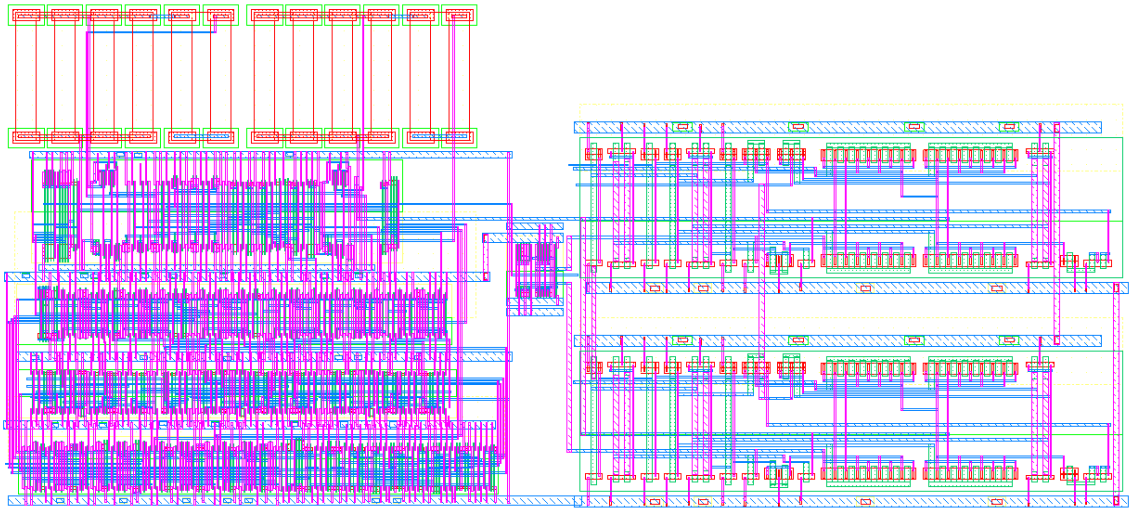

Figure 4.16: The layout of the mixed-mode neuron test structure with memristive synapse.

Chapter 5

Pattern Recognition System

In this chapter, the above described designs of synapse and neuron are used to construct a crossbar based system with online STDP based (supervised) learning. This system is simulated and tested for a pattern recognition application. The proposed system's design is described as follows.

5.1 Crossbar Structure and Operation Scheme

The block diagram of the system built here is shown in Fig. 5.1. It is comprised of two layers of neurons interconnected by memristive synapses, thus forming a crossbar configuration. The input layer of neurons $N_{i_{1\ldots p}}$ produce an input spike pattern depending on the input data. These input spikes lead to current flow in the crossbar synapses, resulting in accumulation in the output neurons $N_{o_{1\ldots q}}$. Later, the winner-takes-all (WTA) logic takes-in accumulated values from the output neurons, to select the neuron with the highest accumulation among them.

Fig. 5.2 shows the details of the WTA block. The operation principle of the WTA is as follows: bit values accumulated in the output neurons are processed individually, with the most significant bit (MSB) having highest priority. If the value in neuron N_{o_m} is $b_{2_m}b_{1_m}b_{0_m}$, where b_{2_m} is the MSB, it will be declared as the winner if $b_{2_m} = 1$ and no other output neuron's MSB is equal to 1, that is, $\sum_{r=1,r\neq m}^{r=q} b_{2_r} = 0$. In case any one of the $b_{2_{\{1..q\}-m}}$ bits are equal to 1, or in case all of $b_{2_{\{1..q\}}}$ are 0, the next lower significant bit b_{1_m} is compared

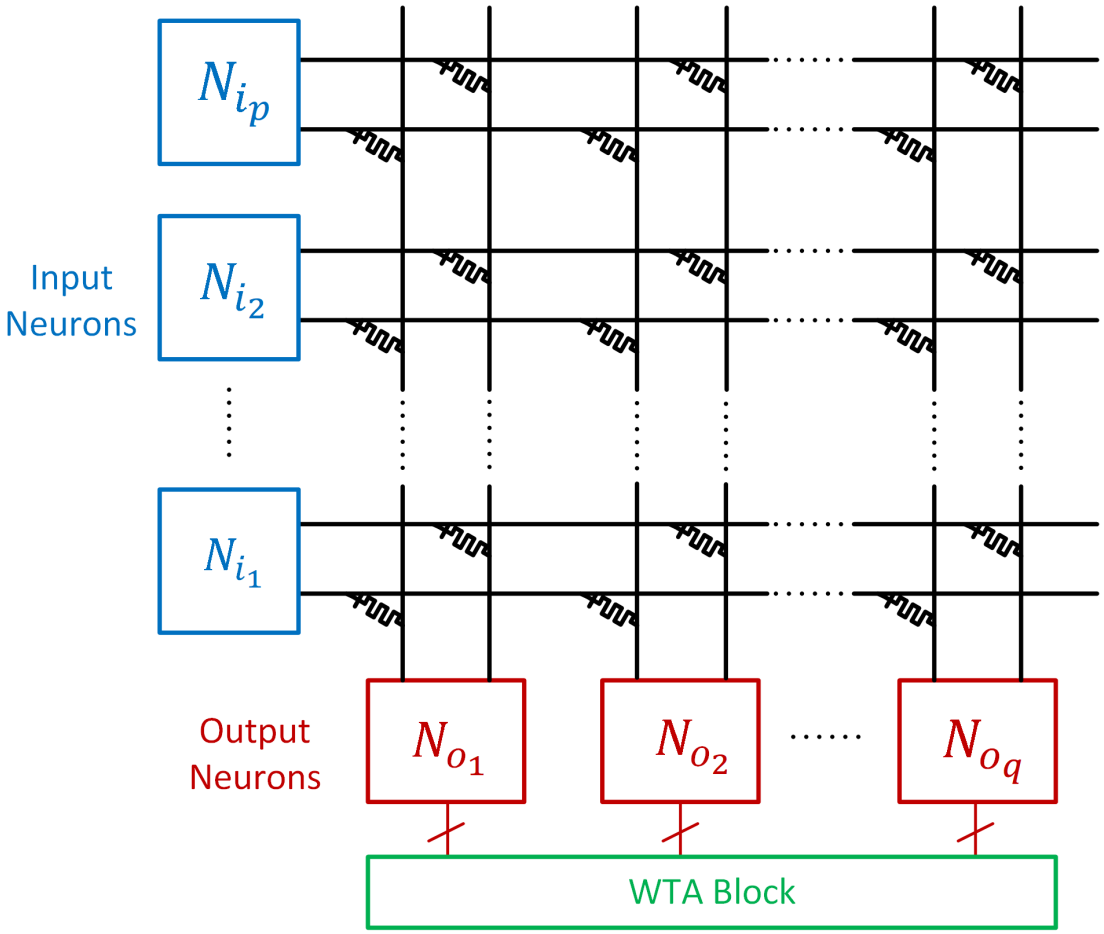

Figure 5.1: The proposed crossbar based system for pattern recognition applications. It may be particularly noted how the proposed synapse is used here at each crosspoint.

with $b_{1_{\{1..q\}-m}}$ in a similar way. Again, if $b_{1_m} = 1$ and in case any of $b_{1_{\{1..q\}-m}}$ is 1 or if all of $b_{1_{\{1..q\}}}$ are 0, the process proceeds to b_{0_m}.

To realize the above described logic, accumulation bits from the output neurons are made to drive some MOS switches in the WTA block as shown in Fig. 5.2. At each bit b_x (x=1,2,3) position, all of these switches' resistances are parallely connected. As described earlier, for decision making at individual neuron level, three cases are to be evaluated: (i) whether all neurons' bits are 0 at that position, (ii) or whether one of the neuron's bit is 1, and (iii) or if more than one neuron's bits has a 1 at that position. Such a global comparison result is combined with each neuron's local accumulation to determine if it was the winner. In order to make such comparison at each bit position in hardware, switches (1X size) at individual

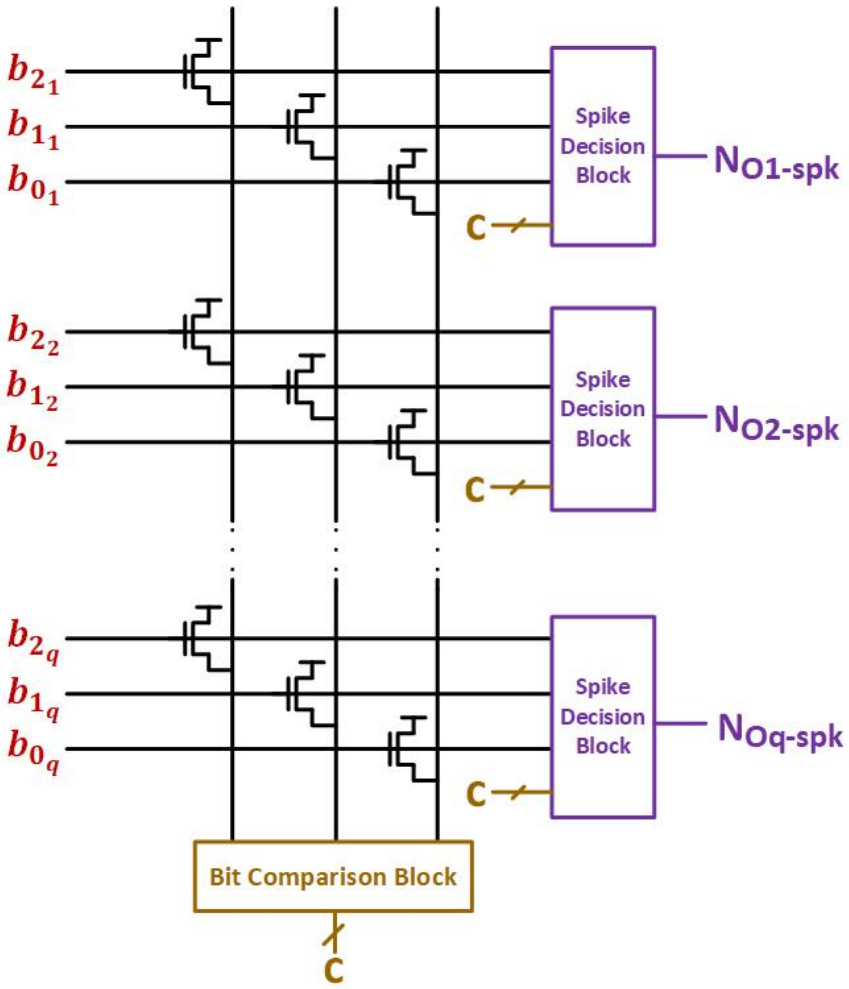

Figure 5.2: The proposed WTA circuit for decision making based on digital bit comparison at each bit position and later using it for neurons' spike decision.

bits drive current into a bit comparison block, shown in Fig. 5.3. Here there are other switches that drive currents in opposite direction and the summation of these currents into the block is used to identify the bit comparison result. The switch Q_1 is 1.5X in size and determines if multiple neurons have a 1. If so, two or more switches will be ON and the total incoming current will be positive and hence *d1* is 0 and it will be equal to 1 if one or none of the neurons have 1 at this position. Q_0 is 0.75X in size and determines whether at least one neuron has a 1. Similarly to above, *d0* is 0 if at least one neuron has a 1 and is 1 if none have a 1. Therefore, after considering the results both the comparisons done above, c_x (x=1,2,3) is evaluated which tells us the number of neurons with a 1. c_x is equal to 1 if

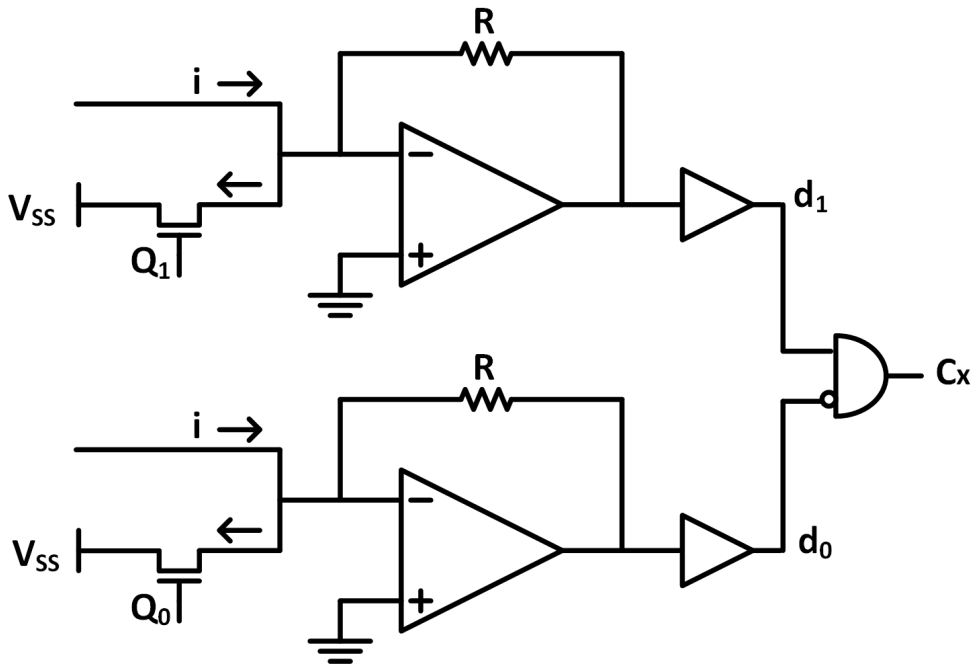

Figure 5.3: Circuit for bit comparison at each bit position.

and only if one neuron has a 1 per column and is 0 if more than one or none have a 1. This bit count information at all the bit positions is input to the individual output neurons' spike decision blocks. Here, a combinational logic (shown in Fig. 5.4) determines if the neuron has to spike (i.e., if it is the winner), as explained earlier: a neuron wins if $c_2 = 1$ and its $b_2 = 1$. If $c_2 = 0$, similar conditions are checked for the next bit position and so on. To validate the concept presented here, the proposed WTA logic was implemented in Cadence Virtuoso and a test case simulation's results are presented in Table 5.1. 10 digital values (coming from accumulations in 10 neurons) are input to the logic and it can be seen that the logic correctly 'chose' the highest value and did not choose anyone when more than one neurons had the highest value.

Depending on the results provided by the WTA logic for each neuron (about whether it is the winner or not), the signal N_{o-spk} in Fig. 5.2 controls the feedback switches (S3 & S4) at each neuron. Since only one neuron (among all the output neurons) will spike (if at all there is a spike), only the 'winning' neuron will generate and apply a feedback spike (leading to STDP based learning in the crossbar synapses). Moreover, since the spike propagation is

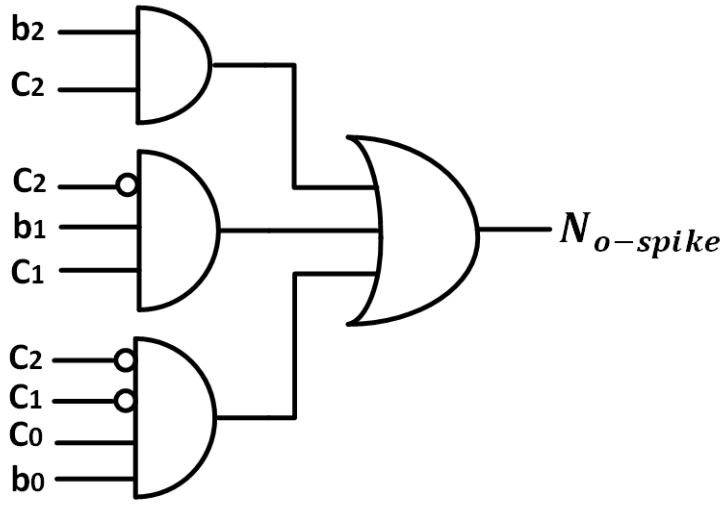

Figure 5.4: The output neuron's spike decision circuit.

gated locally at each neuron (by N_{o-spk}), a single spike generator block can be used by all of the output neurons, thus helping reduce the hardware costs and enhancing the scalability of the system at large.

Table 5.1: The WTA logic's simulation results under two conditions (1) one output neuron has the highest value (left) (2) more than one with the highest value (right).

Neuron	Accumulation	Spike Decision
N_{o_1}	001	0
N_{o_2}	010	0
N_{o_3}	001	0
N_{o_4}	100	0
N_{o_5}	110	1
N_{o_6}	011	0
N_{o_7}	001	0
N_{o_8}	010	0
N_{o_9}	100	0
$N_{o_{10}}$	011	0

Neuron	Accumulation	Spike Decision
N_{o_1}	001	0
N_{o_2}	111	0
N_{o_3}	001	0
N_{o_4}	100	0
N_{o_5}	010	0
N_{o_6}	011	0
N_{o_7}	001	0
N_{o_8}	111	0
N_{o_9}	100	0
$N_{o_{10}}$	011	0

5.2 Training

To simulate and validate the crossbar based system (Fig. 5.1) and its operation as described above, its behavioral modeling was done using Python language (see Appendix B). In this work, a handwritten digits data set from the UCI Repository [31] is used with this system as the application. It is comprised of 8x8 size input patterns representing the digits 0 to 9. These patterns have input elements with values between 0 to 16. This dataset used here contains 3823 digit patterns for training and another set of 1797 patterns for testing. Because the inputs are 8x8 sized patterns and comprise of 10 classes, the system has 64 input and 10 output neurons, one each for every input matrix element and a digit class, respectively. In this work, the matrix values have been downsampled to the range 0 to 7 such that 0-1 is encoded as 0, 2-3 as 1 and so on, as shown in Fig. 5.5.

For training the crossbar weights, the input data is converted into a spike train following a temporal rule on when the spikes must occur, as shown in Fig. 5.5. The downsampled values for the input data are used to calculate the delays (in multiples of clock periods) in producing the corresponding input neuron's spike. The input neuron receiving a value of x spikes after x clock periods ($t_i + x$) as shown in that figure. Also, a supervised learning model is used here, implying that class labels for inputs are read into the simulator and the corresponding output neuron (for the input class) is made to spike, in order for learning to happen through STDP in the crossbar. Whenever an input is provided to the system, the corresponding output neuron is made to spike before the training time period begins (t_{i-1}) and after it ends (time t_{i+1}). Note that the spikes used in this work contain four different positive voltages occurring over a span of four clock periods, implying that STDP can be facilitated at a synapse for up to four clock periods of time difference between the neurons' spikes. Hence, based on the first spike of the output neuron, the synapses with inputs of the range 0-3 will be depressed, with the 0-input leading to most magnitude of depression. Similarly, synapses with inputs of the range 4-7 will be potentiated due to the second output spike (occurring at the end of training phase). It is noticeable here that the 1st spike of the output neuron does not influence potentiation and vice-versa because the time separation is too large for an overlap to occur.

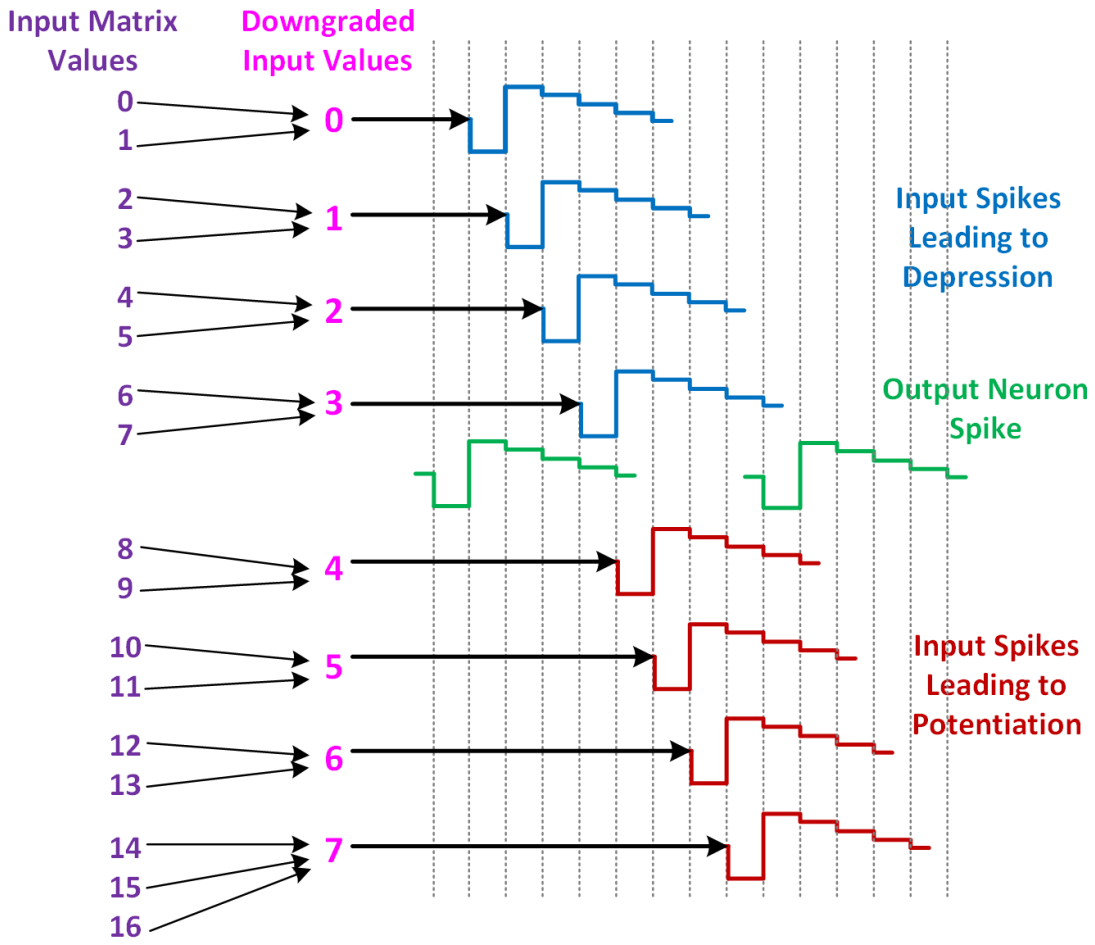

Figure 5.5: The downsampling of input values and the consequent spike encoding used at the input neuron layer.

For the purpose of learning operation on the crossbar system, the whole 3823 patterns in the training set are used. The crossbar synapses are initialized to zero weight at the start of the training ($M_p = M_n = HRS$). With the progress of learning, synapses are subject to potentiation and depression cycles. The *learnt weight* attained by a synapse in due course is dependent on the relative number of potentiation and depression cycles that it undergoes. For example, synapses whose inputs were predominantly in the 4-7 range, undergo more potentiation than depression, and they eventually attain a high positive weight. The contrary occurs to synapses whose inputs were predominantly in the 0-3 range. Also, for synapses whose inputs were about the same from both the ranges, their weight settles in between

the either extremes. Fig. 5.6 demonstrates this property wherein the learnt weights of the synapses on all columns of the crossbar are plotted as digit patterns.

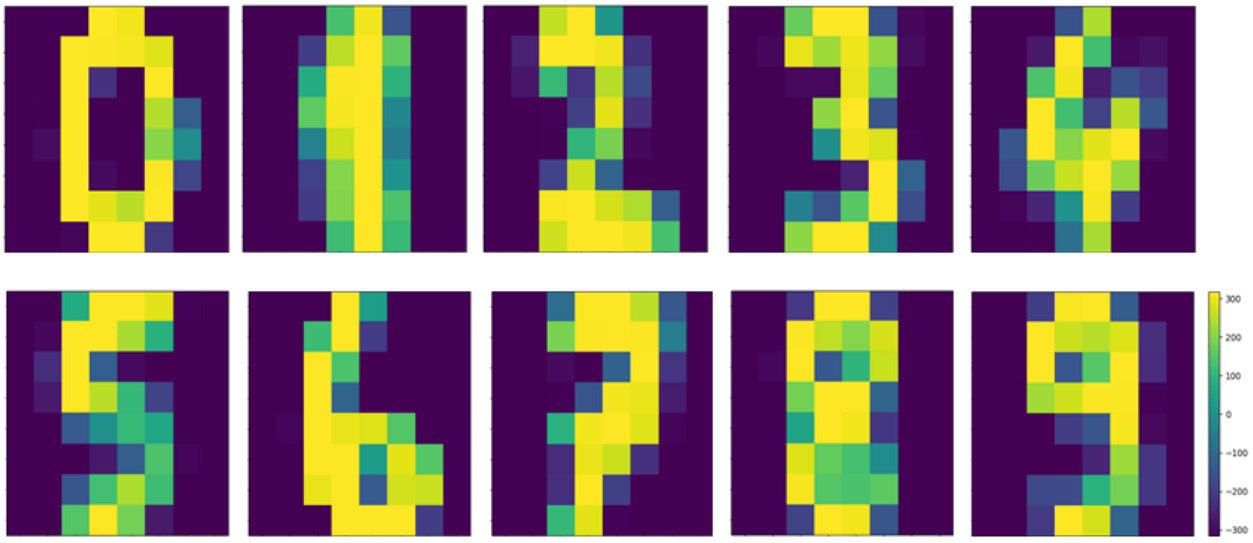

Figure 5.6: The weights attained by the crossbar synapses after the learning phase. The colorbar (at the bottom right) displays the conductance scale in μS.

5.3 Testing

The crossbar after being trained as described above must be 'tested' for accuracy using the 1797 input patterns from the testing part of the dataset. When inputs are applied, currents flow through each column, leading to accumulation in the corresponding output neurons. Based on the accumulation therein, WTA logic here plays the role of an arbiter to determine the 'winner'. In case only one output neuron has the most accumulation, it is considered the winner and the corresponding label is checked against the input label to validate the testing under consideration. In case there is a mismatch, or if the WTA block was unable to choose a single winner neuron, then the particular input test case is said to failed the test. Fig. 5.7 shows the graph for accuracy of the crossbar system plotted against the training epochs. The recognition accuracy here reaches its maximum value by the end of one training epoch and it remains saturated thereafter. The accuracy of classification seen here is comparable to the 83% reported in earlier works [87] using the same dataset and a similar system design.

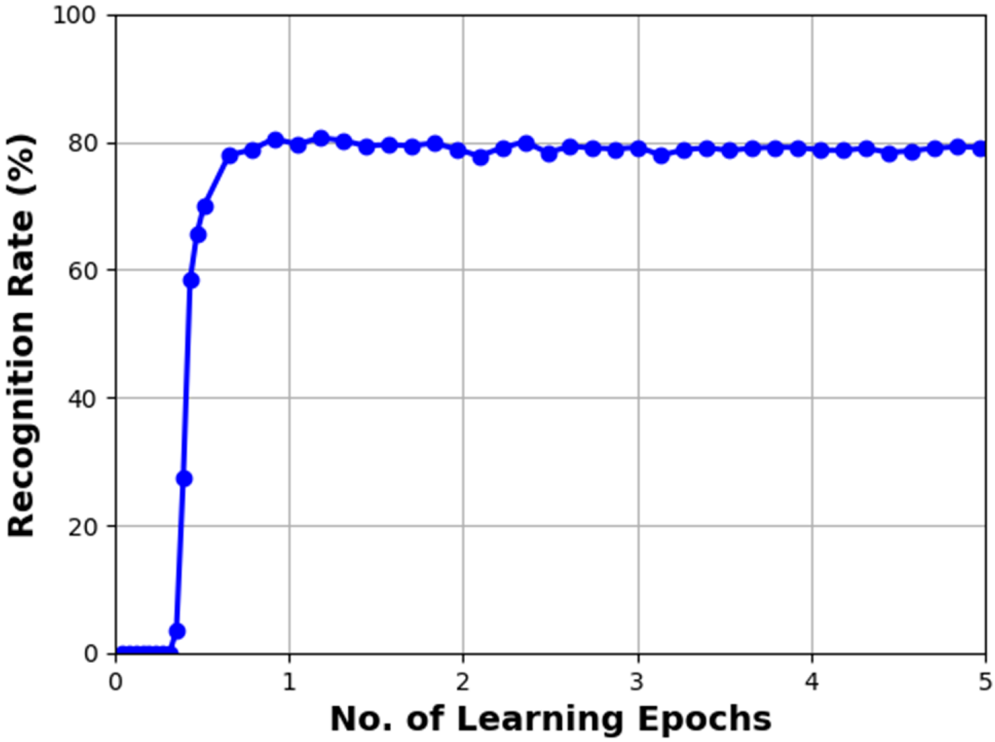

Figure 5.7: The accuracy of recognition versus training epochs .

Moreover, in this work only a mere STDP based supervised learning rule on a single layer crossbar was used, wherein the final weights achieved in the crossbar are directly dependent on the STDP property used. This lets us analyze the robustness of the proposed circuits at the system level, as will be demonstrated in Section 5.4. Additionally, the confusion matrix for this system and this dataset is shown in Fig. 5.8. It is evident from this figure that the inference of digits was mostly correct (boxes along the diagonal have high values). Also, the commonly misinterpreted bits were 8 and 1, which may be caused by the similarity in the conductance of the central synapses of both these patterns as seen in Fig. 5.6. Also, the digit 1 was confused with 2 and 5 with 9, which can be similarly be understood as being due to the similar placement of the high weight synapses in these patterns.

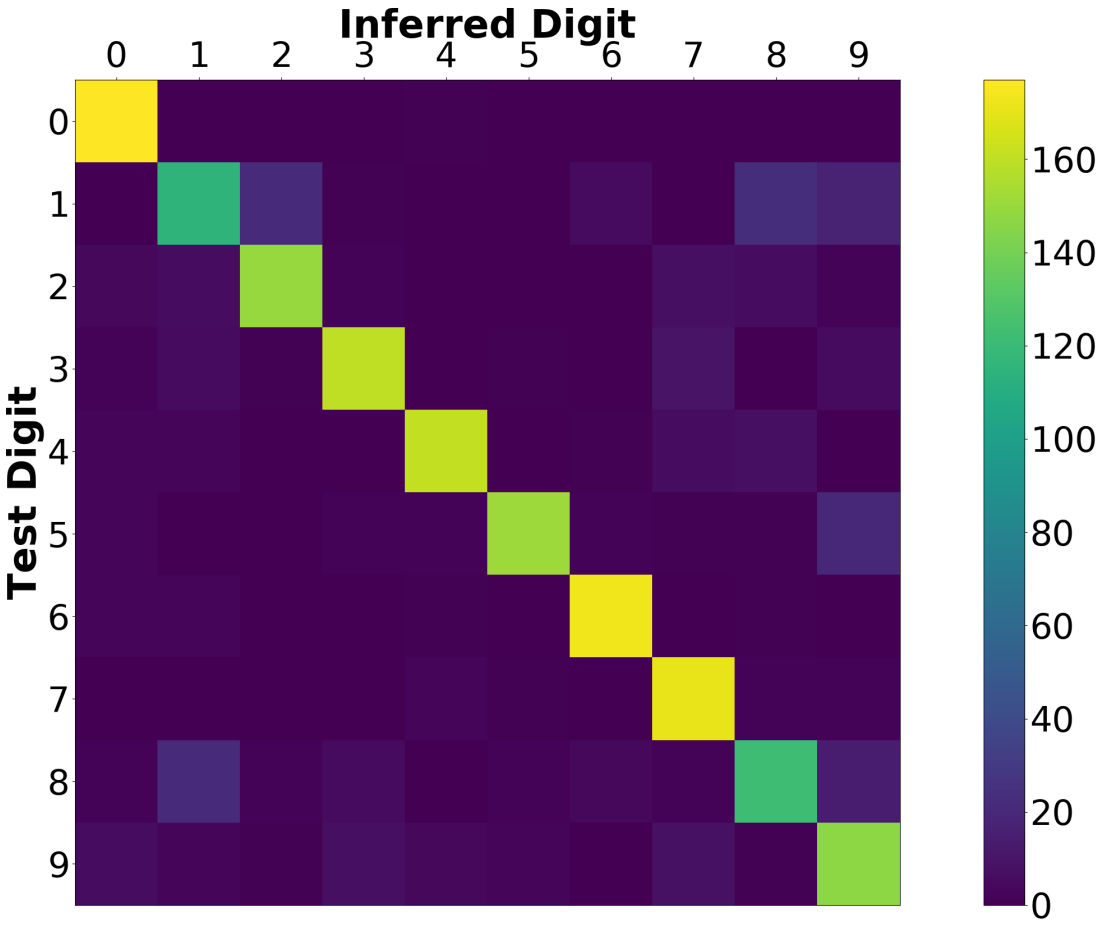

Figure 5.8: Confusion matrix obtained during the testing.

5.4 Performance Analysis

In this section, the system described above has been simulated under a variety of practical non-ideal situations and the results are presented to demonstrate the robustness of the proposed synapse and the mixed-mode neuron designs under such situations.

5.4.1 Impact of Clock Frequency

As seen in Fig. 3.7, the weight update magnitude during STDP in this work is a function of the frequency of the clock used. It was shown there that use of a high frequency clock results in small ΔG and vice-versa. This influence of clock frequency on the ΔG during STDP and

thereby its impact on the learning of this pattern recognition system was studied here and is shown in Fig. 5.9.

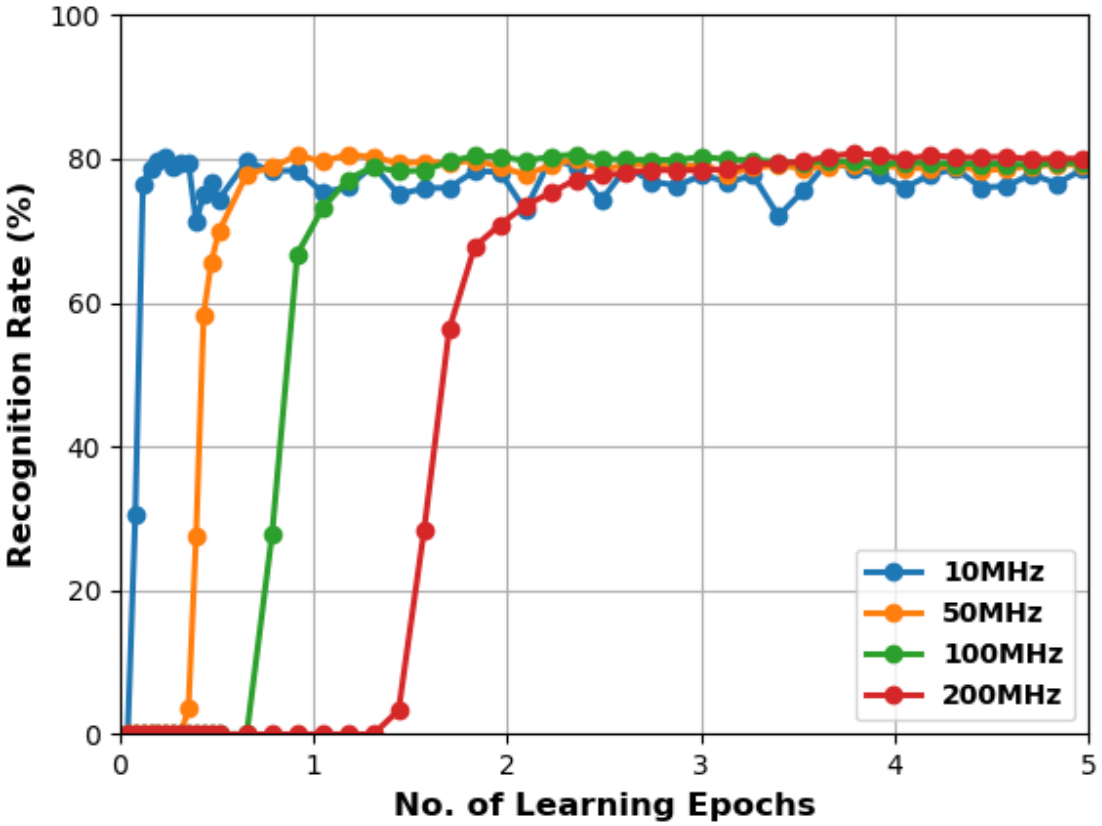

Figure 5.9: The crossbar learning behavior as a function of the clock frequency.

Fig. 5.9 shows that at high frequencies, the systems takes more epochs for learning, which is understandable due to the diminished ΔG during STDP. This indicates that the learning rate can be increased by lowering the frequency. However, if one reduced the frequency too much, the learning behavior does not saturate smoothly. This is because low frequency leads to large ΔG during STDP resulting in reduced granularity of learnt weights.

Based on the above discussion, we can say that there is a trade-off between learning speed of the system and learnt weights' precision. Therefore, this implies that a suitable design frequency must be chosen for this kind of system. In this work, 50MHz is adopted because the system attains its maximum accuracy after the first epoch in this case and

curve remains smooth thereafter. This discussion also implicitly points us to another critical device property: resolution of resistance. It can be seen that a low resolution of weights leads to poor precision and granularity of weights during learning. Therefore, it is needed that devices are engineered to provide as many stable resistance states as possible between their extremes. The techniques presented here enhance this capability by providing the designer a circuit level control over the resolution. By choosing an appropriate clock frequency, the designer can control and/or tune the increments in weights.

5.4.2 Impact of Device Switching Asymmetry

Switching Speed Asymmetry

Section 3.3.3 demonstrated that switching speed asymmetry at the device level can lead to uncontrolled changes in STDP and hence a crippled STDP behavior. This effect is shown at the system level in Fig. 5.10 and Fig. 5.11.

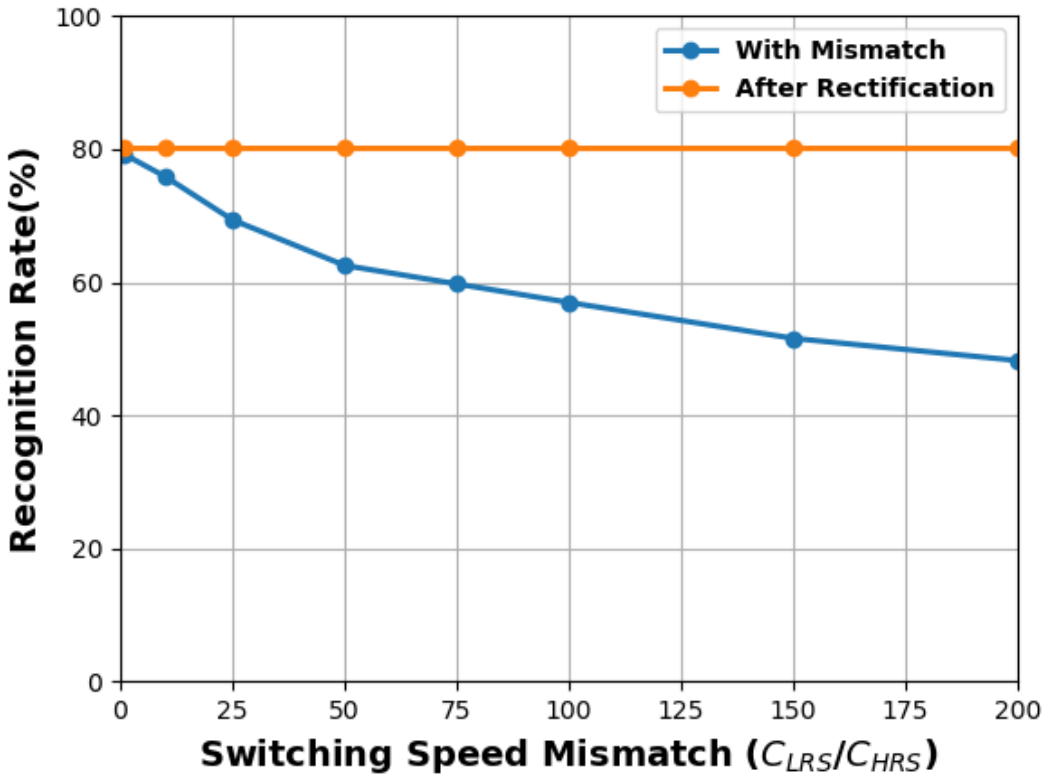

Figure 5.10: The effect of switching speed asymmetry on the system's recognition accuracy.

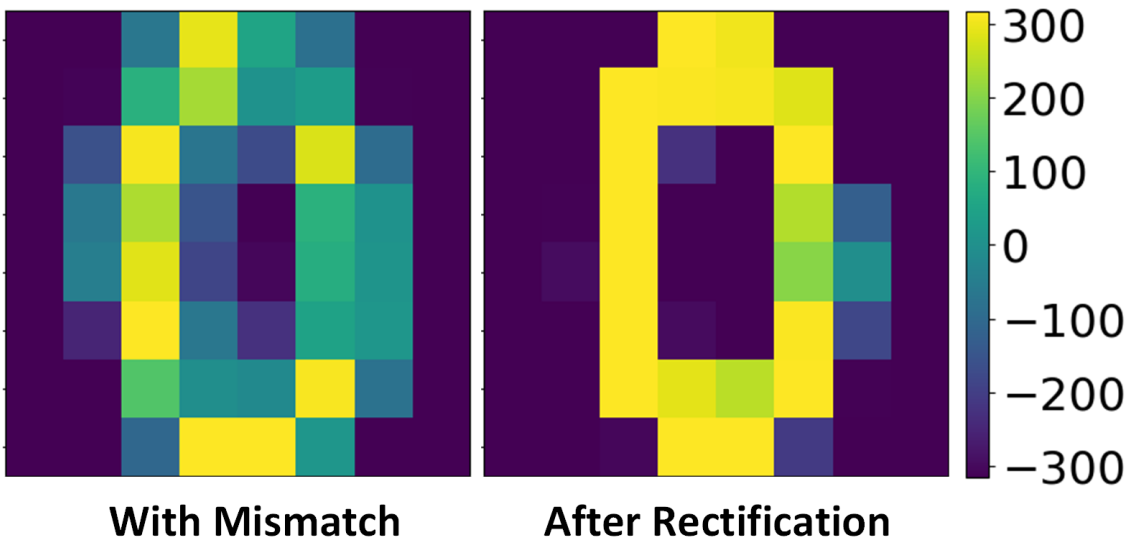

Figure 5.11: An example of the impact of switching speed asymmetry on the learnt weights in the crossbar. The pattern '0' is shown here.

This figure indicates that as the switching rate in the device gets more and more asymmetric, the classification accuracy deteriorates accordingly. As Section 3.3.3 also demonstrated, by employing duty modulated signals in the feedback spikes, the effect of asymmetry in switching can be mitigated. This is shown at the application level in Fig. 5.10. In addition, Fig. 5.11 shows how the learnt synaptic weights in the crossbar will be hampered in the presence of this effect and how they are rectified after applying the remedy technique described above.

Switching Threshold Asymmetry

Apart from switching speed asymmetry, the other significant and commonly occurring deviant behavior is the switching threshold asymmetry. The change in memristance during switching, ΔM, is dependent on the overdrive $V - V_t$ applied to it. Therefore, if V_t in both the directions are not equal, the overdrive in one direction will be less than the other, leading to less change in that direction. This leads to unequal ΔM in both directions. This can have adverse effect on the synapse's STDP characteristics and thereby on the system level learning. This is shown in Fig. 5.12. Here, both the types of switching asymmetries are simulated and their rectification is applied independently and also in conjunction. It may be

noticed that a combination of both these non-idealities proves very detrimental to system's functioning and that it has failed quickly due to their impact.

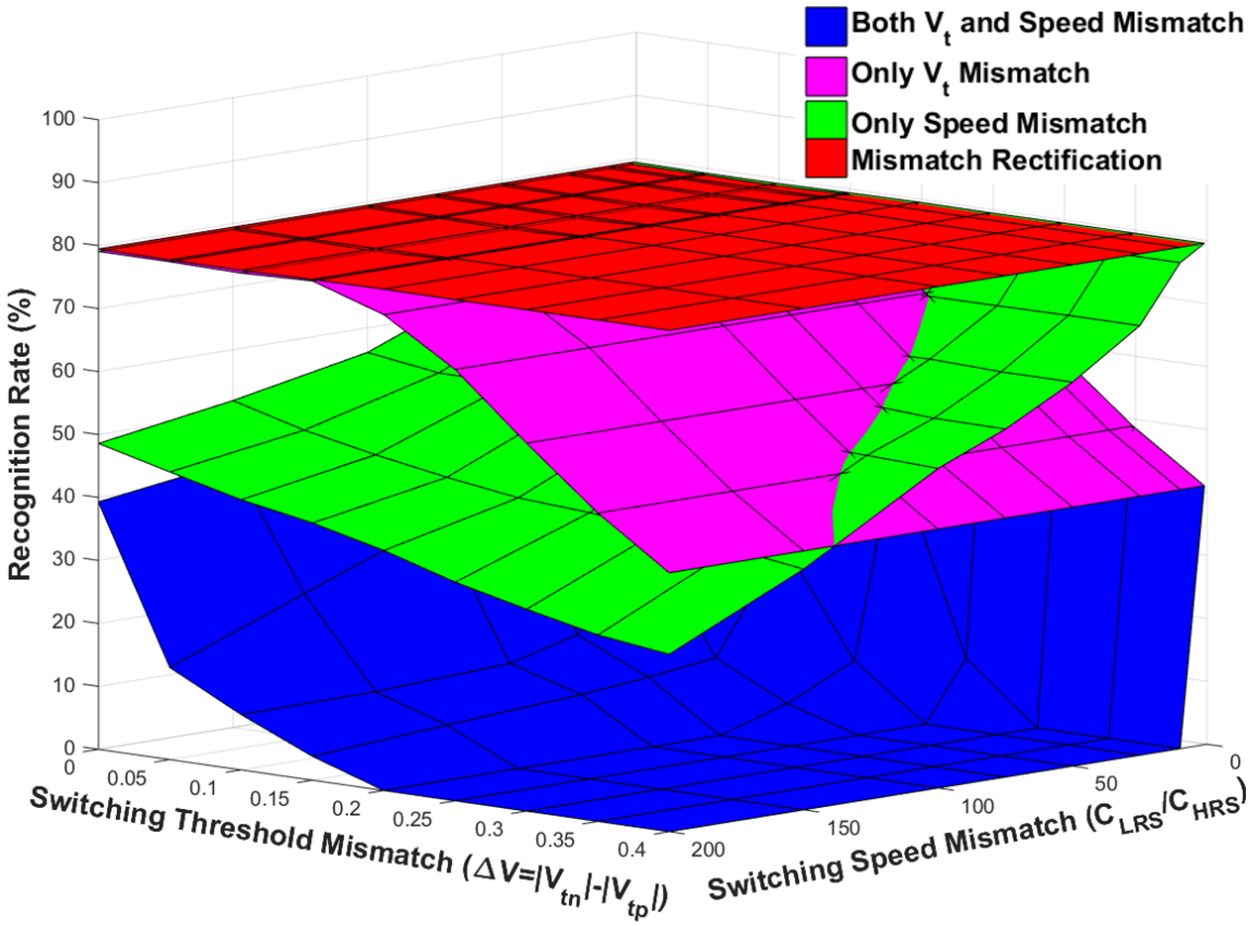

Figure 5.12: The impact of switching speed and threshold asymmetry on the system's accuracy.

The neuron spikes used in this work comprise of discretized time steps and also the voltage values. This provides the flexibility of carefully choosing and/or tuning these parameters as needed for the application at hand. This implies that in the face of switching threshold asymmetry, one can compensate for the reduced overdrive by tuning the spike voltage used for learning. This is shown in Fig. 5.13. In this illustration, $\mid V_{tn} \mid > \mid V_{tp} \mid$, which is typically the case. So the effective negative bias across the device needs to be raised to compensate for the high V_{tn}. During the learning phase, a negative bias is applied (on M_p for depression and

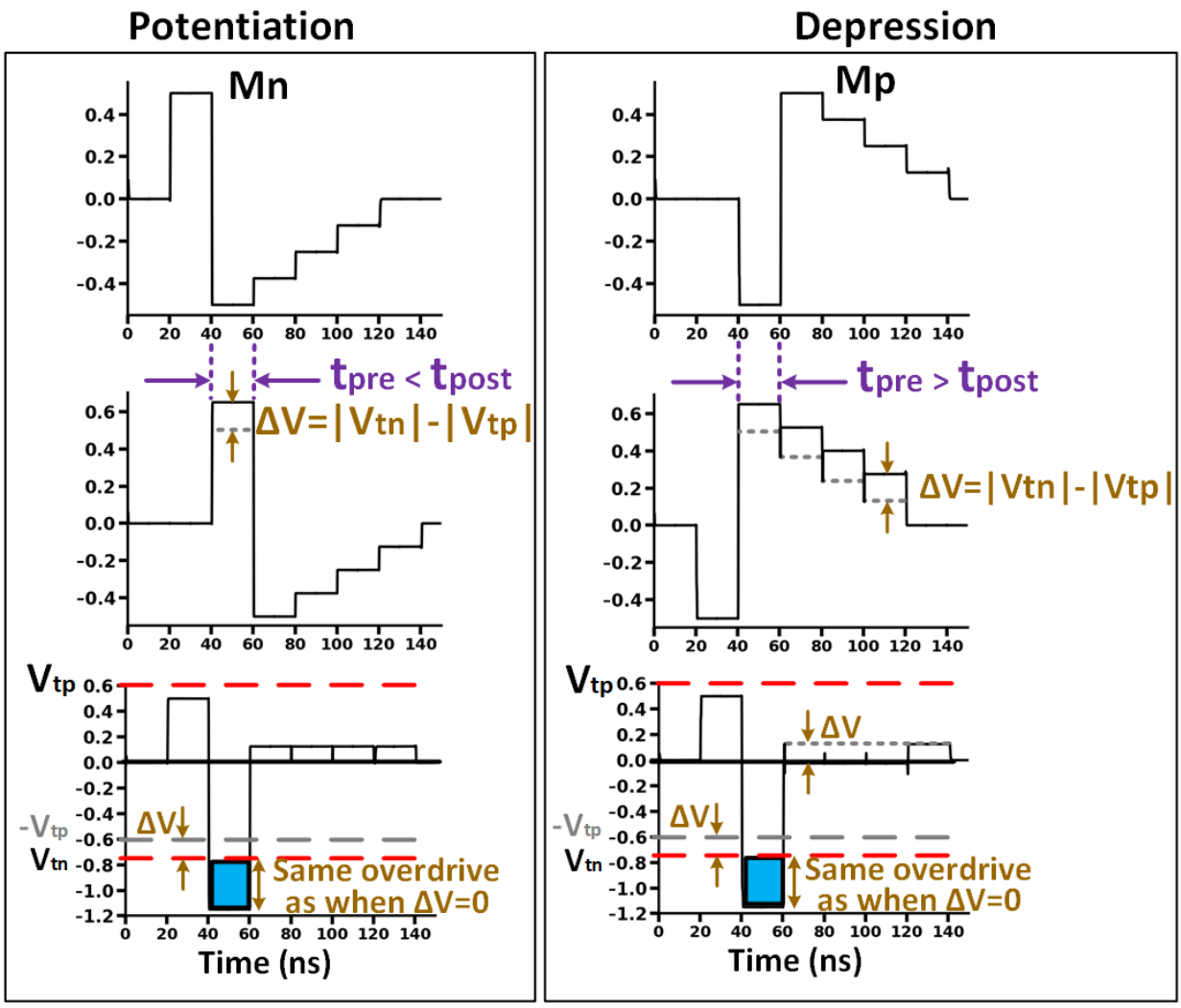

Figure 5.13: Feedback spike voltage modification to remedy threshold asymmetry effects.

on M_n for potentiation) by virtue of applying a negative potential from pre-neuron and a positive one from post-neuron's feedback spike. Therefore, as a compensation technique, the positive voltage in the feedback is raised. In addition to this, in the presence of both types of switching asymmetries, both of the remedy techniques described individually above may be adopted together to mitigate such effects. Such a simulation result is shown in Fig. 5.12. Here it is demonstrated that by using the combined compensation techniques, accuracy can be restored to its original value which would otherwise go down all the way to zero (as seen in the blue surface plot).

5.4.3 Impact of Memristor Device Change

It was illustrated earlier in Chapter 4 that with a 'fixed/rigid' neuron that is specifically designed to work for a given memristive synapse, a re-design is needed before it can be used with a new device type, else the neuron will fail to 'adapt' to new memristors. This property is illustrated at the system level in Fig. 5.14.

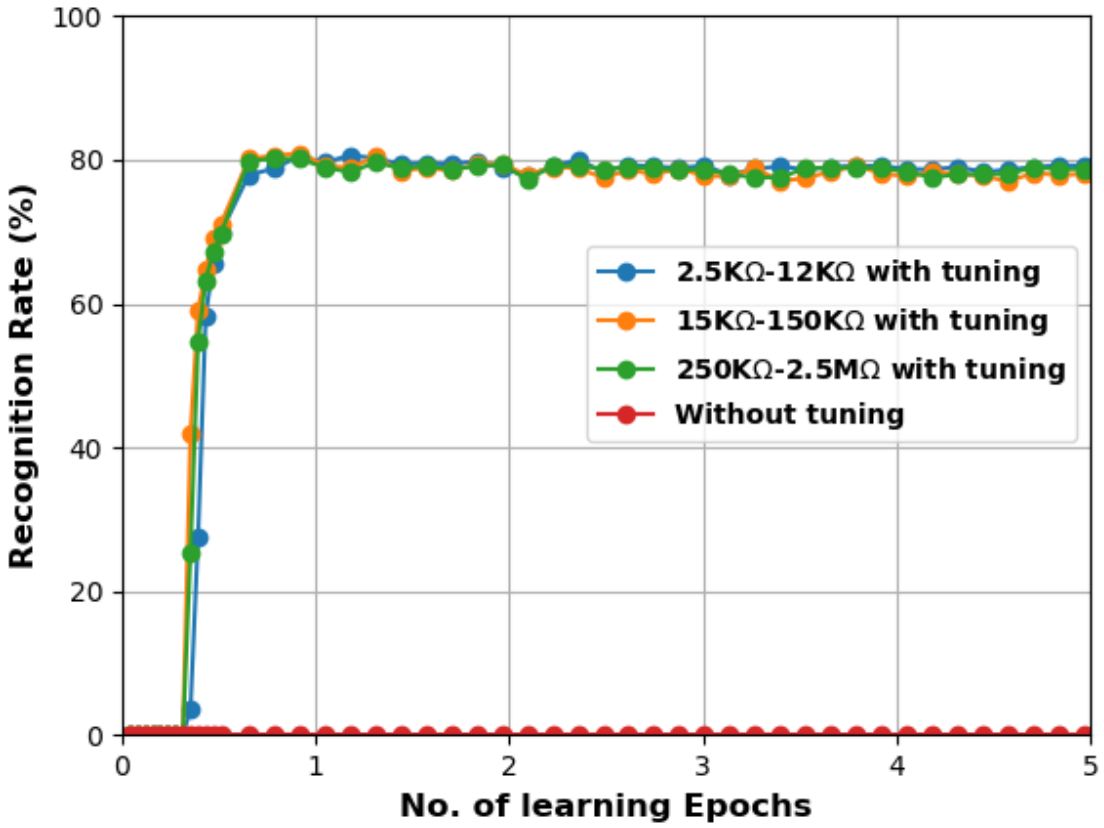

Figure 5.14: The accuracy of the system is retained for various memristor types used, owing to the tunability of the proposed neuron. Here, 'without tuning' implies a simulation run with a neuron designed for a given device ($2.5K\Omega - 12K\Omega$ here), but used with other two memristor types.

The used neuron here was designed for the $2.5K\Omega - 12K\Omega$ memristor and simulated for the other two memristors without tuning its accumulation rate. The figure indicates that the system failed to achieve its task for this case. Later, the neuron's accumulation rate was

tuned to 'adapt' for each new device type that was used and it can be seen that the task was accomplished successfully and the accuracy was restored. This was made possible by the use of a tunable transimpedance amplifier in the proposed neuron in this work. This underscores the fact that the proposed neuron can be tuned on-chip to adapt to a variety of memristors in the back-end-of-line without having to change the core silicon design, thus saving prototyping costs for this technology and its development process.

5.4.4 Impact of Switching Endurance

Reliable, repeatable and consistent switching behavior between the LRS and HRS of the memristor is a desired quality. However, with repeated switching cycles occurring to the device, it tends to lose its ability to reproduce its initial behavior. This is referred to as the switching endurance, which typically manifests itself as a degradation of the LRS and HRS values of the memristor [116, 22, 79]. This often culminates in the diminishing of the switching window HRS/LRS of the memristor [22]. This property was modeled at the system level by changing the LRS and HRS values from their original ones, with an increase for LRS and a decrease for HRS. Fig. 5.15 depicts that for the case of a rigid neuron, the accuracy gradually falls. However, for the tunable neuron proposed in this work, its accumulation rate can adjusted in accordance with the changing LRS and/or HRS and consequently the accuracy can be revived as seen in that figure. Also, in this analysis, at 45% degradation of LRS and HRS of the $2.5K\Omega - 12K\Omega$ device, the HRS/LRS ratio is less than 2 and hence the accuracy starts to roll off. Therefore, this analysis illustrates that the proposed neuron can be tuned to cope with the detrimental impact of limited switching endurance of the memristor devices.

5.4.5 Resolution of Mixed-Mode Neuron

As shown earlier, the neuron encodes its input current into an 'n' bit digital value and performs accumulation/threshold comparison in the digital mode. The influence of the choice of this encoding resolution on the accuracy is shown Fig. 5.16. The accuracy starts to saturate beyond n$\geq$3 and is equal to 80%, 84% and 84.75% for n=3, n=4 and n=5,

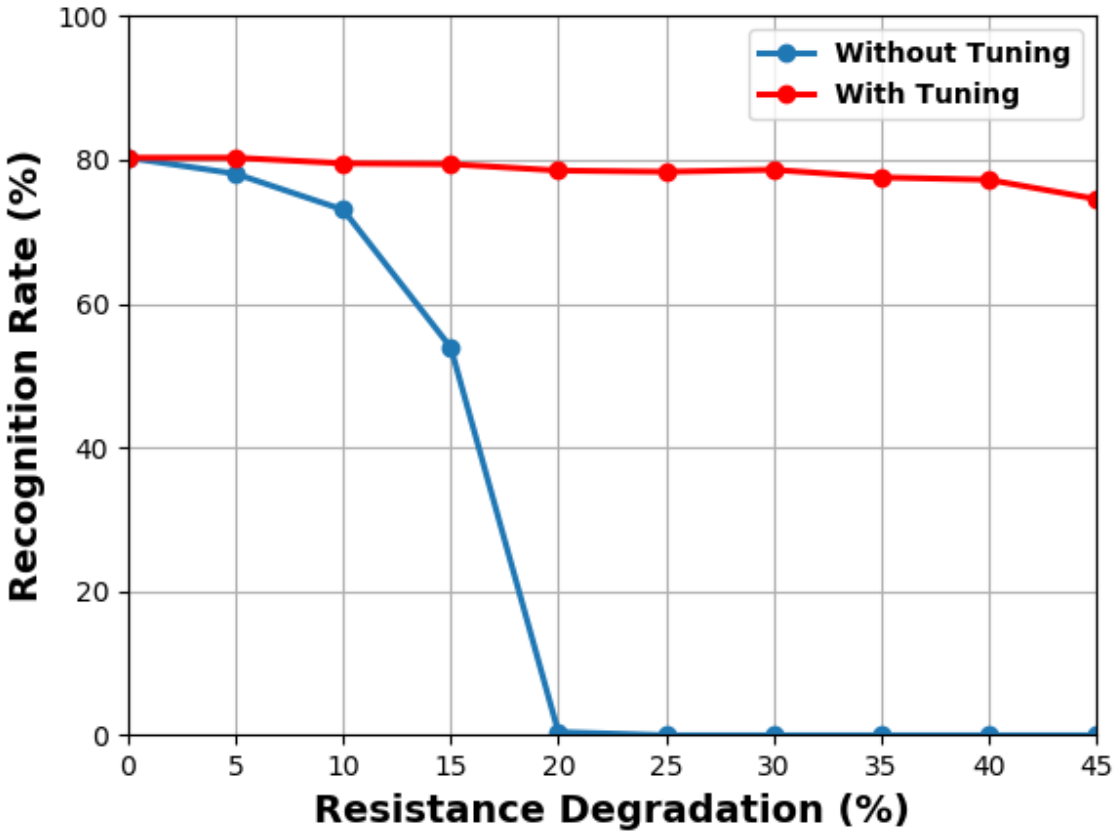

Figure 5.15: The accumulation tunability of the neuron used to remedy the impact of the device's switching endurance (shown here as % degradation from their original value).

respectively. In order to keep the hardware costs to a minimum, n=3 was chosen in this work since there was not much improvement in accuracy beyond it. Moreover, in [87], 83% recognition accuracy was reported for the same dataset. Hence, the accuracy obtained here is competent with that in literature, given that the primary focus here is on robustness.

5.4.6 Area Overhead and Energy Consumption

The physical design (layout) of the neuron was done using a 65nm kit and the area estimation of the layout is shown in Table 5.2. The proposed mixed-mode neuron occupies less silicon

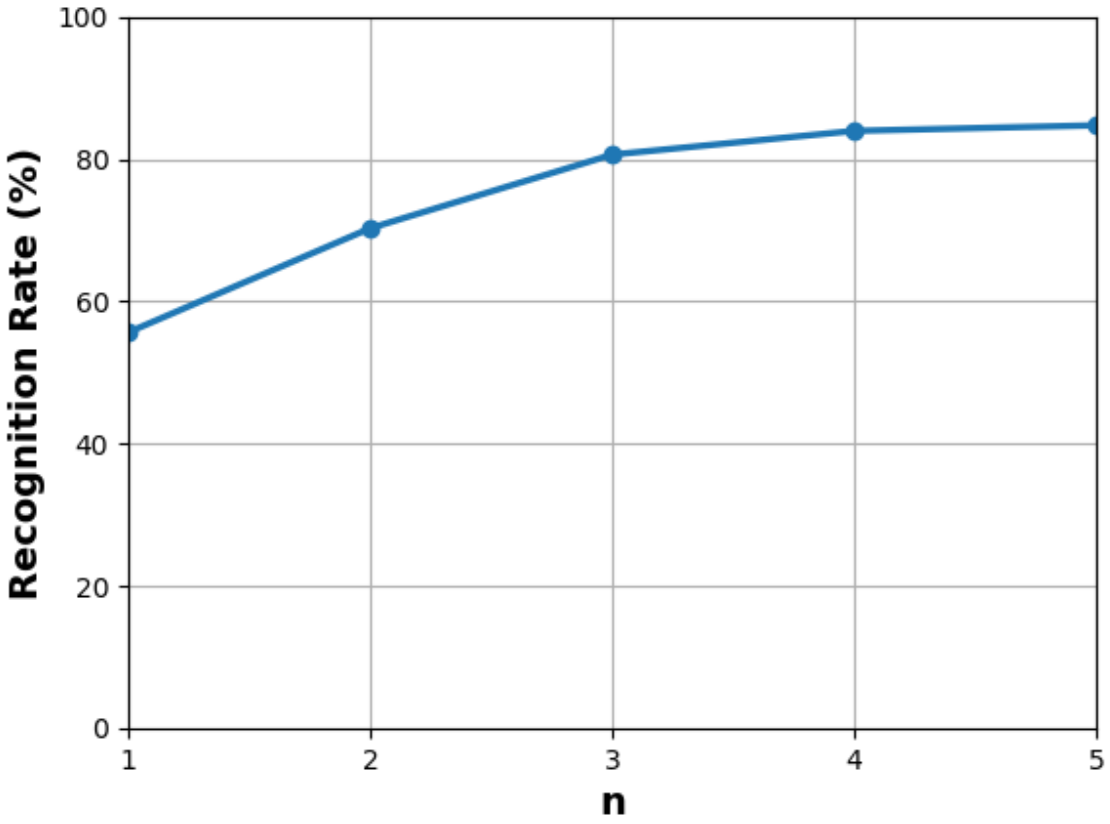

Figure 5.16: The impact of neuron's resolution 'n' on the accuracy.

area than a conventional IAF neuron (employed in [19]) that consists of hardware-costly capacitors (of upto a few pF) [91].

Table 5.2: Area overhead for the neurons' physical design

	This Work	Integrate and Fire Neuron [91, 19]
Layout Area (μm^2)	$\sim$ 1400 (65nm)	$\sim$ 2000 (65nm)

Moreover, as the crossbar increases in size for other applications and/or if the used memristor has lower resistance, column currents increase leading to larger integrator capacitors in the IAF neuron. Additionally, if the neuron's layout changes significantly across

devices and/or applications, the overall floorplan of a system might also need considerable changes. Therefore, the mixed-mode neuron proposed in this work offers a small and fixed layout design without the need for change across implementations.

To evaluate the energy cost of the proposed mixed-mode neuron, its energy consumption per spike per synapse is shown in Table 5.3 and is compared to the other reported values in the literature. It can be noticed that this neuron consumes lesser energy that most others in the literature. Also, the IAF neuron in [19] used a synapse with resistance close to the device Mem_1 here (2.5kΩ-12kΩ) and had more energy consumption, thus proving the energy efficiency of the proposed neuron.

Table 5.3: Energy consumption of neurons during spiking

	Energy (pJ/Spike/Synapse)	Synapse condition used
This work	1.05	250kΩ-2.5MΩ
This work	2.4	15kΩ-150kΩ
This work	8.1	2.5kΩ-12kΩ
[19]	23.07	2kΩ-10kΩ with digital spike
[115]	9.3	1000 synapses with 1MΩ each
[39]	36.7	70Ω-670Ω
[17]	11-0.1	1kΩ-1MΩ

5.4.7 Other Design Considerations

The spikes utilized in this work are discrete in terms of the voltage levels they consist of, which helps boost the noise robustness and scalability of a system. When larger crossbars are used for bigger neuromorphic applications, the parasitic resistance of the crossbar interconnects (used for the columns) increases. To reduce this resistance impact and the possible voltage drops, the interconnects could be made wider. But this will also lead to an increase in the parasitic capacitance due to the interconnect which is in addition to the diffusion-related capacitance contributed by the MOS devices connected to this interconnect. This implies that in using an analog spike (such as those based on exponentially damping shapes)

there will be a charge sharing issue between the parasitic capacitors of the crossbar and the spike generating circuit. This problem is circumvented by the use of a capacitor free spike generation concept here. The use of constant voltage pulses and careful on-chip tunability of the spike voltage levels provides the advantage of being able to tackle system level issues post-fabrication too.

Additionally, the WTA logic proposed here introduces less area overhead per neuron at the output layer (making it scalable). Since only one of the output neurons spikes, the spike generation circuit may be shared amongst all of them. Apart from being scalable, the system design presented here also helps alleviate sneak currents issue [124, 62] in the crossbar owing to its half-bias scheme. The power rails used here have a VDD (positive)-VSS (negative) scheme. As shown in Section 3.2, the proposed neurons' outputs are held to a mid-rail voltage (GND in this case) when there is no spike. Similarly, the column nodes are biased at a 'virtual GND' by the output neurons. This implies that the crossbar is biased to mid-rail potential in an 'idle' state, thus alleviating sneak currents.

Chapter 6

Memristor-Based Reservoir Computing

In this section, the crossbar based system design presented in the earlier section is applied to build a bigger neuromorphic system implementing the concept of reservoir computing.

6.1 Background of Reservoir Computing (RC)

Feed forward neural networks have been thoroughly developed and used in machine learning tasks. However, they are primarily capable of approximating static (non-temporal) input data. For working with temporal data, recurrent neural networks (RNNs) are better equipped as they consist of feedback connections too. However, training these networks is an involved process and can be very slow [93]. Reservoir computing is a paradigm that taps the advantage offered by RNNs in terms of data processing while overcoming the need for complex training process. As Fig. 6.1 illustrates, an RC system comprises of three layers, of which the 'reservoir' receives the inputs $u(t)$ from the input layer and transforms it into a higher dimensional space. The reservoir layer's output (indicated as $x(t)$) is connected to the 'readout layer' via the weights denoted as W_{out}. The readout layer performs the final stage of processing and its output is the result of classification.

During the operation of the RC system, only the output weights W_{out} are modified by a learning procedure, whereas the weight matrix of the reservoir itself (which is an RNN) is

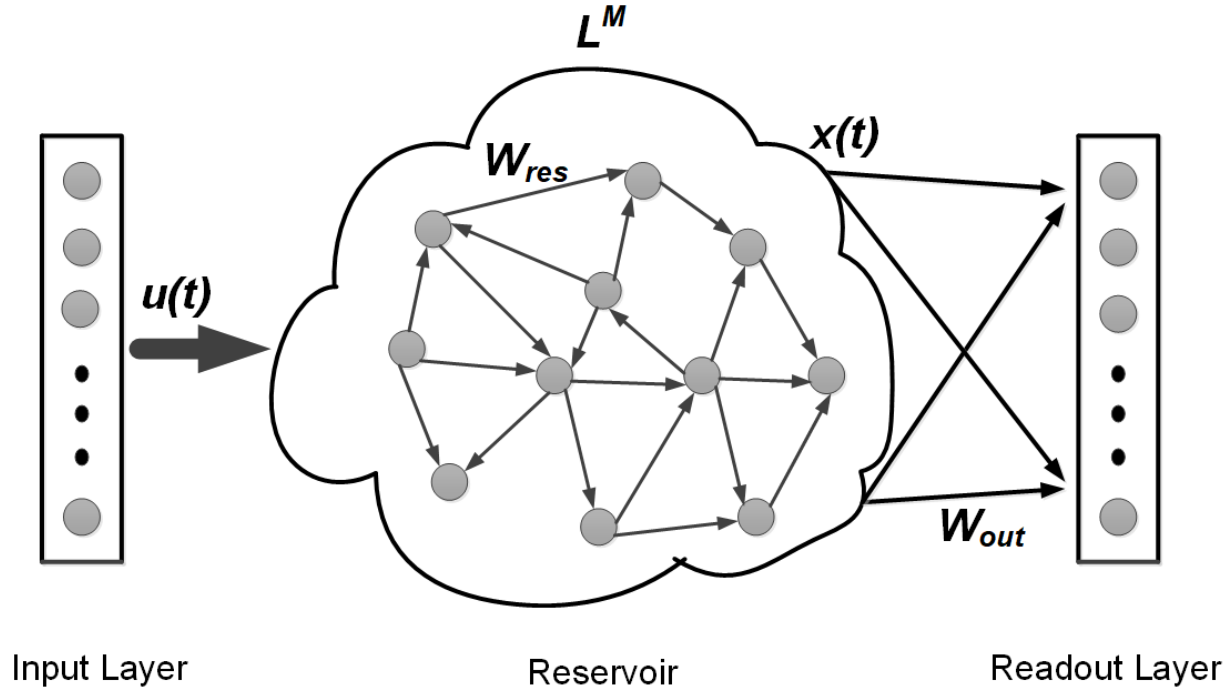

Figure 6.1: The illustration of a reservoir computing system consisting of three layers: input layer, reservoir layer and the readout layer.

left unchanged after they are randomly initialized and implemented. Hence, the primary advantage of the RC framework is that the RNN part of the system is left untrained, thus circumventing the need to go through a computationally intensive and slow training process. The concept of reservoir computing when implemented with neural networks can have two kinds of realizations: liquid state machines (LSMs) and echo state networks (ESNs). ESNs comprise of artificial neurons performing the computation, whereas LSMs are more biologically inspired and employ spiking neurons with recurrent connections in a way that makes it more likely for spacially closer neurons to be connected.

An added advantage of the RC framework is that it can be realized using a wide variety of physical systems. Such a physical implementation can provide fast processing of input information while minimizing the learning cost involved. The following section summarizes the existing literature on physical RC.

6.2 Related Work on Physical RC

Any system that offers rich dynamics and has the capability of non-linearly mapping the inputs to a higher dimension can be utilized to realize RC. Reservoir computing has been implemented using a variety of physical systems offering rich dynamics. These systems include: neural network based systems [92, 95, 112], delayed dynamical systems [10, 55], cellular automata [119], photonic systems [111, 110], spintronics based systems [108, 70] and mechanical systems [109, 37]. A more detailed review of this is presented in [106]. The concentration in this work is on building RC systems based on spiking recurrent neural networks in the neuromemristive domain, which are suitable for VLSI implementation for applications such as embedded systems [102].

A range of literature has presented purely CMOS based RC designs. In [118], an echo state network was implemented on an FPGA and was thus digital CMOS based. In [78] liquid state machines completely based on digital designs were presented and an architecture for their implementation was proposed that was reconfigurable. In [121] a digital LSM with bio-inspired learning in the readout layer was presented and a performance analysis was presented on this for various design and operating conditions in [45]. A power optimized digital system similar to the earlier one was presented in [113]. However, these systems are all digital CMOS circuits based, which are area intensive. They typically need a lot of silicon real estate owing to the bulky circuits they need such as registers and/or other memory arrays.

Since memristor is a nanoscale device, memristive alternatives for reservoir computing have been explored. One of the first works to dicuss the possibility of a memristive RC system was [53]. In this work, a network of memristors was built to act as the reservoir. A regular mesh configuration of memristors was proposed in [16] and was shown to be tolerant to variations. RC was shown physically using memristor devices in [29]. All of the above techniques had used some topology consisting of a network of memristors as the reservoir. They did not explore memristive neural networks as the reservoir.

In [65], epiliptic seizure detection application was shown using ESN employing memristor based neurmorphic circuits. A similar work was presented in [52], where two versions of it

were considered: digital and mixed-signal. In [36] RC based on memristive crossbars was proposed wherein two crossbars were used for weight implementation. One of the crossbars here realizes the reservoir layer weights, whereas the other is for readout layer weights. An RC based on ESN with 'least mean square' approach for learning in the readout was presented in [114]. However, all of the so far mentioned works utilized ESNs for RC systems, but not the bio-inspired (spiking neurons based) LSMs. In [101], a memristor based LSM was presented. However, the implementation here was area intensive because it used 'heavy' circuits such as ADCs and DACs.

6.3 Memristor-Based RC

In this work, the focus is on building a simple memristive neuromorphic circuit based LSM for RC realization. The reservoir layer here is built with spiking recurrent neural networks and the readout layer comprises of a memristive crossbar layer where the synapse is the bi-memristor synapse presented in Chapter 3. A striking feature of the work here is that the weights in the reservoir layer are based on memristors being programmed to their extremes. This implies that analog (intermediate) weights between extremes are not considered, thus making the system simpler and more reliable. In the readout layer, which is the only place where training takes place, it is based on the STDP scheme presented earlier. This scheme is simpler to realize compared to the other complex approaches in the literature as explained earlier. Additionally, this training in the readout layer is shown to be robust to device level switching issues, owing to the robust circuits employed here.

6.3.1 Memristive Neuromorphic Circuits Based LSM

As explained earlier, the reservoir in this work is a memristor based spiking recurrent neural network (SRNN). However, to determine the optimal SRNN network, several hyperparameters of the network such the number of neuron and synapses and their interconnections must be determined. To simplify the search process for the best parameters, an evolutionary optimization (EO) based genetic algorithm has been employed here [86]. This EO algorithm works as follows: it initially generates a 'population' of networks based

on some starting hyperparameters chosen by the user. Based on how good these generated networks are in performing the given application, they are given a 'fitness score'. A set of crossovers and mutations are carried out on the best performing networks to generate the next generation of networks and this process continues. The final goal of this algorithm is to find a network topology that performs well as a reservoir on the given application.

This optimal SRNN produced by running the EO algorithm is used as reservoir and provided with the inputs ($u(t)$) for the given application. The output spikes from the reservoir constitute the inputs to the readout layer. These spikes lead to current flow in the synapses of the readout crossbar and accumulation takes place in its output neurons. As with the pattern recognition task in Chapter 5, learning here in the readout layer is through supervised STDP approach and a WTA scheme is utilized during testing to calculate the accuracy of classification.

6.3.2 Framework for Implementation of Memristor-based RC

In this subsection, a generic hardware framework for the implementation of the proposed memristor based reservoir computing system is proposed. Using this framework, any given network topology for the reservoir system may be implemented. The RC system as presented in Fig. 6.1 can be divided into three parts: (1) Input layer, (2) Reservoir layer and (3) Readout Layer. The hardware framework for each of these is discussed as follows.

Input Layer

The input layer design here is based on the input encoding scheme used. In this work, a 'binning' based encoding scheme has been used [94]. This input encoding works as follows: in a given dataset, each data element has certain 'attributes' that represent it. Each of these attributes has a certain range of values that it can assume. By choosing 'n' number of bins, the range of values attainable by the attributes is partitioned into n divisions. A total of n neurons are earmarked in the input layer for each input attribute. For a given input value of an attribute, depending on the subdivision it falls into (in the range of possible values), the corresponding input neuron (among the available n) is made to spike. Therefore, for a

total of (say) m input attributes in a given dataset, if the number of bins is n, there must be a total of $m*n$ input neurons, with n neurons allocated per attribute. The block diagram for the implementation of this input scheme is shown in Fig. 6.2.

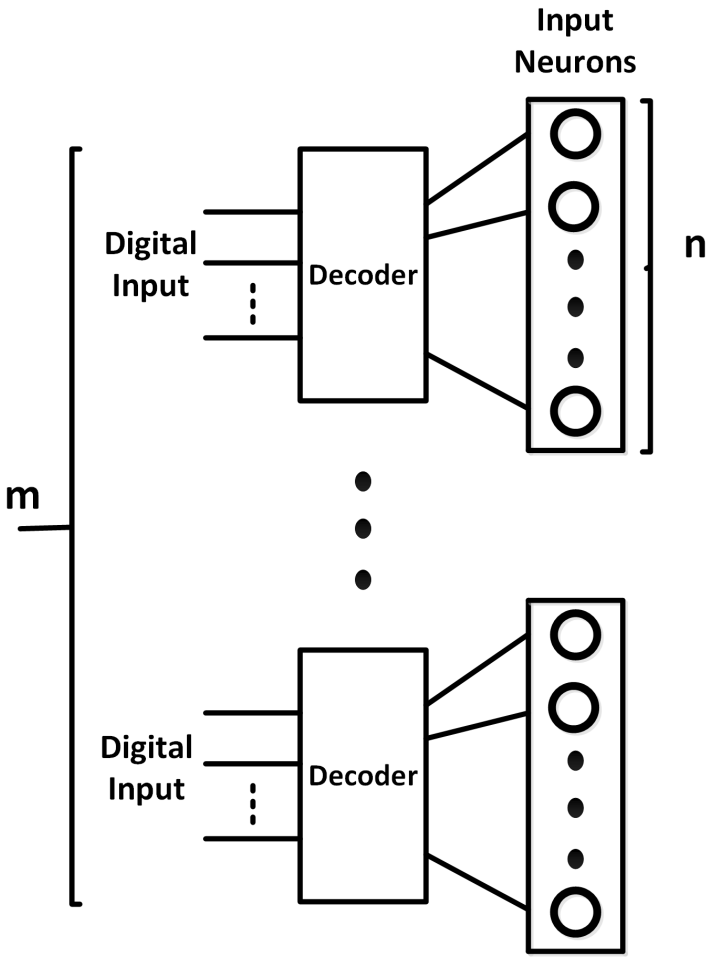

Figure 6.2: The block diagram for a generic framework for input layer implementation. Here, m is the number of input attributes in the dataset and n is the number of bins chosen.

As can be seen from Fig. 6.2, the input neurons are divided into groups of n neurons, each of them allocated for one attribute of the input dataset. There are m copies of such groups for m input attributes. It is here assumed that the inputs are available in a digital encoded format (after processing by a digital computer). Each input data value is encoded into a digital value (based on its value and the number of bins). This digital value is input

to the decoders above. The decoder triggers the corresponding input neuron among the n and hence results in an input spike.

Reservoir Layer

To implement the reservoir layer, a framework known as the memristive dynamic adaptive neural network array (mrDANNA) [19] is proposed here, as shown in Fig. 6.3. This architecture consists of mrDANNA cores, with each of them containing a neuron and several synapses. These synapses may be programmed to a desired weight. By connecting several cores together, connections can be established between neurons with user defined synaptic weights at the input of each neuron. Thus, such connections between cores leads to the realization of any given spiking neural network topology. This can help implement a given reservoir configuration generated by algorithms such as EO.

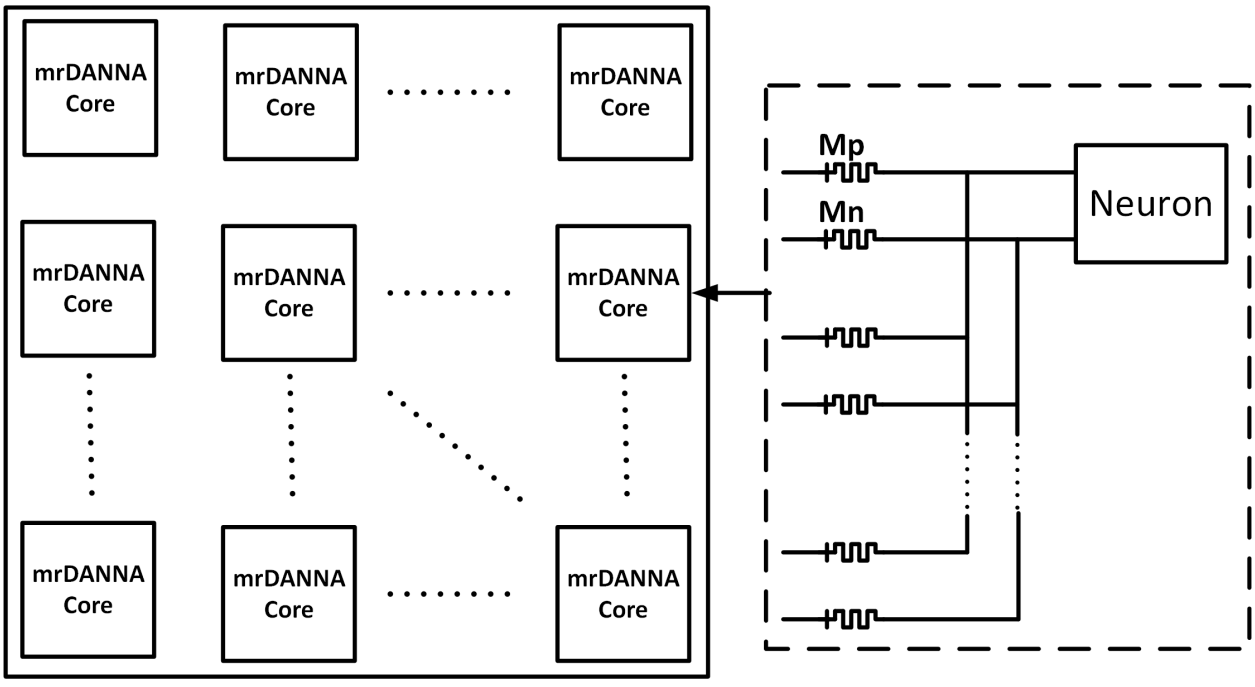

Figure 6.3: The mrDANNA architecture suitable for implementing a given reservoir network topology.

Readout Layer

The readout layer is realized with the memristive crossbar based on STDP learning as described in Chapter 5. This system receives inputs from the reservoir. Supervised STDP based online learning has been used here in this readout layer. For testing purposes, the WTA block acts as the arbiter to decide the 'winner' neuron. The readout layer is shown in Fig. 6.4.

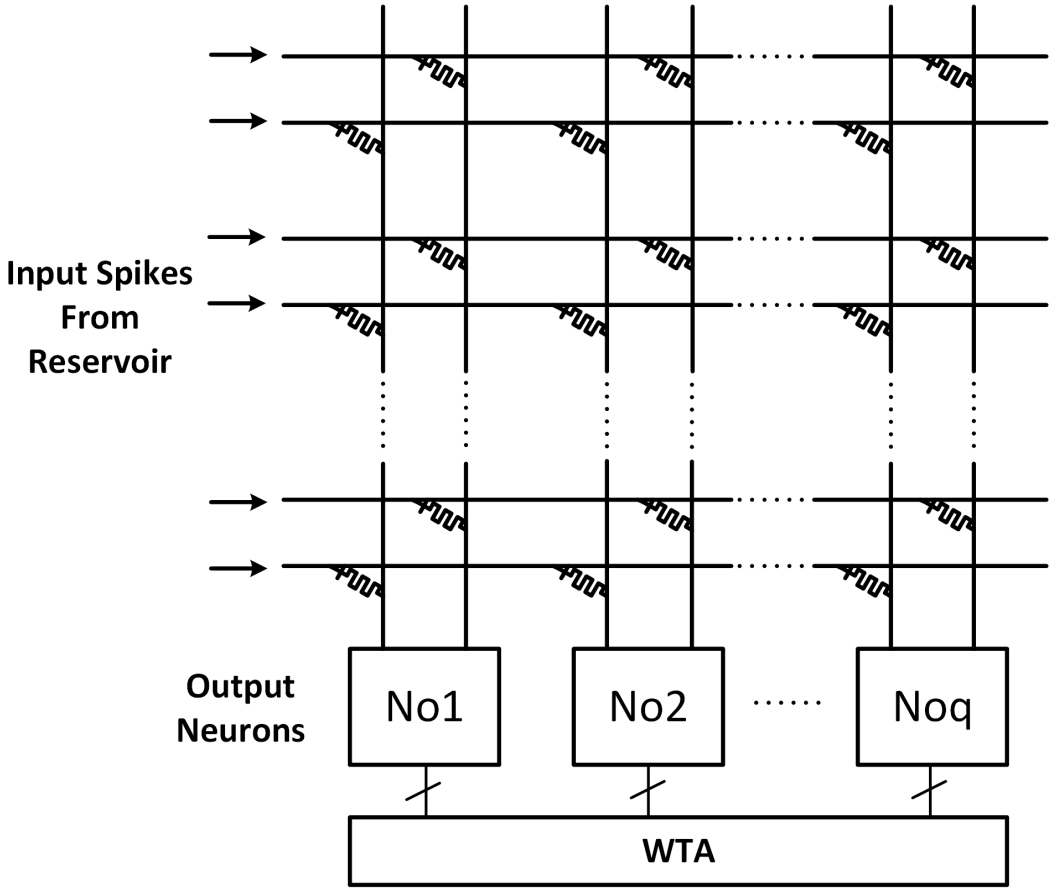

Figure 6.4: The memristive crossbar structure used here as the readout layer.

The Complete System

The complete system comprising of the above blocks is shown in Fig. 6.5. It may be noted here that system is reconfigurable and is amenable for implementation of any given topology of a reservoir computing system.

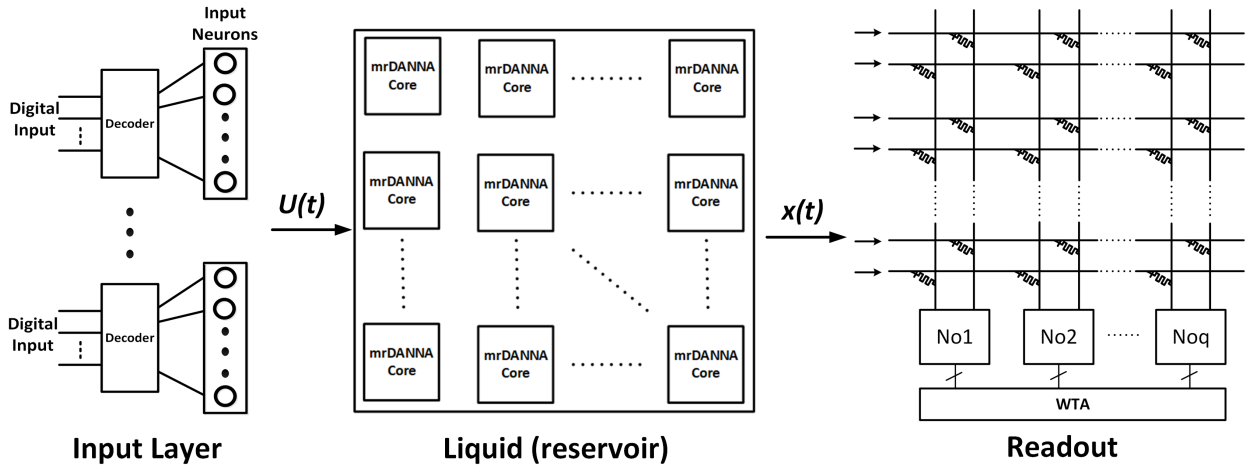

Figure 6.5: The block diagram of the complete reservoir computing system.

6.3.3 Simulation Results

For simulating the reservoir system, a high level simulation model for the network and the system was developed using C++ and Python. 'Wisconsin Breast Cancer (WBC)' and 'EEG' datasets have been used here. The WBC dataset (from the UCI machine learning repository [31]) comprises of 9 input attributes that describe various features of cell nuclei such as radius, perimeter, area, etc. Fig. 6.6 illustrates this dataset where all the data is plotted on the same figure. It may be noted from this figure that the data is not linearly classifiable because the various data points for the two classes are entangled. This dataset was provided to the software based model for the LSM proposed here and the training was performed on the readout layer. The classification accuracy for this dataset as training progresses in the readout layer is shown in Fig. 6.7. It may be noted that the RC system here attains an accuracy of about 93.7%.

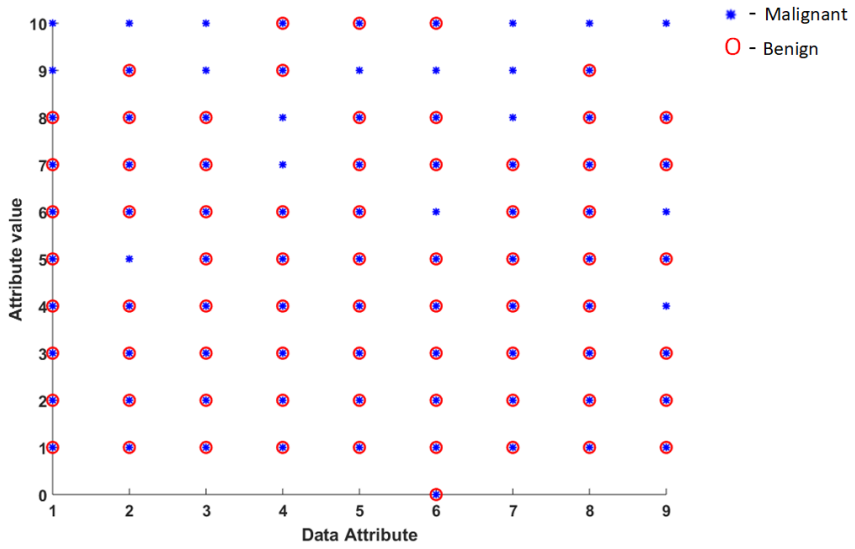

Figure 6.6: The WBC dataset's input attributes. It can be seen that the data is non-linearly classifiable.

Similarly, the EEG dataset (which is time-series based) was also used with this setup. This dataset consists of data representing activity in the brain, which can be used to identify the health condition of the brain. This dataset comprises of data for healthy as well as an epileptic brain. The RC system was used here to differentiate between these two classes of input data. The results are shown in Fig. 6.8.

6.3.4 Effect of Device Switching Asymmetry

In the proposed RC system, learning only occurs in the readout layer. Hence, any device-level switching issues are expected to adversely impact the recognition accuracy of this system. This was studied for both the applications above and is shown in Fig. 6.9. It can be seen that while switching time and threshold independently impact the learning here, their combination has a significant impact. Here, switching rate mismatch of 50X and a threshold mismatch of 0.2V is chosen, which is the mid-value of the range chosen for the analysis in Fig. 5.12. Also, the techniques presented in Section 5.4.2 for switching asymmetry mitigation were applied here and the results are shown in Fig. 6.10.

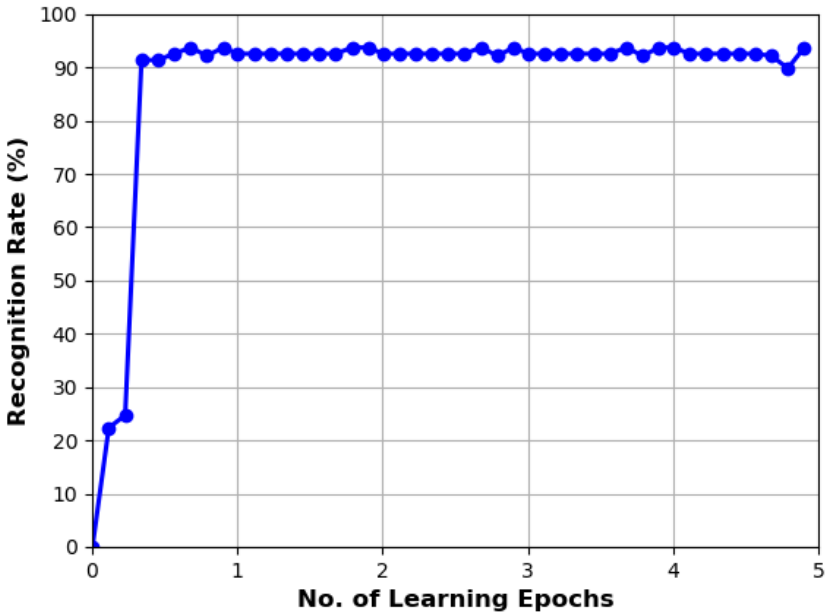

Figure 6.7: The accuracy of classification attained for the WBC dataset as training progresses in the readout layer.

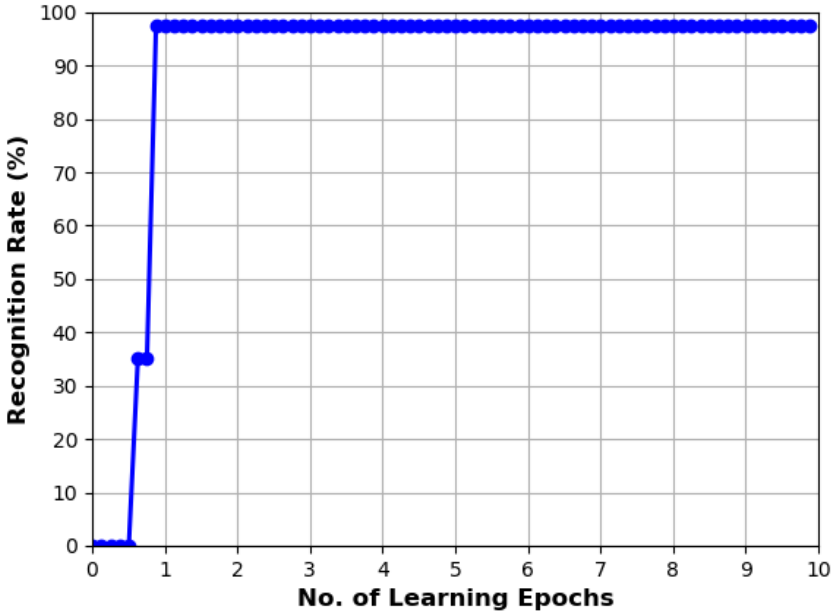

Figure 6.8: The accuracy of classification attained for the EEG dataset as training progresses in the readout layer.

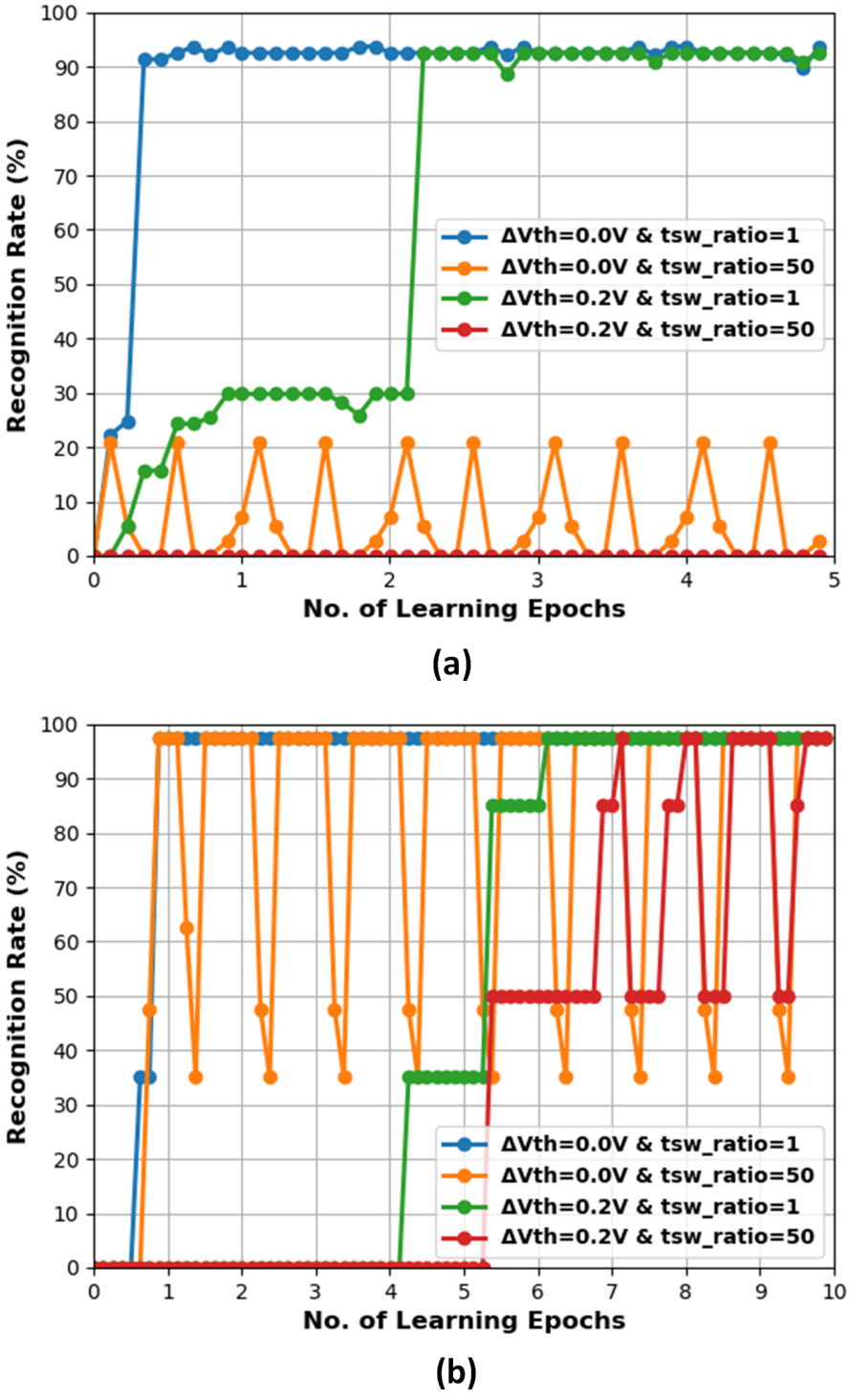

Figure 6.9: The impact of device switching speed and threshold asymmetry on the readout layer learning for (a) WBC dataset (b) EEG dataset.

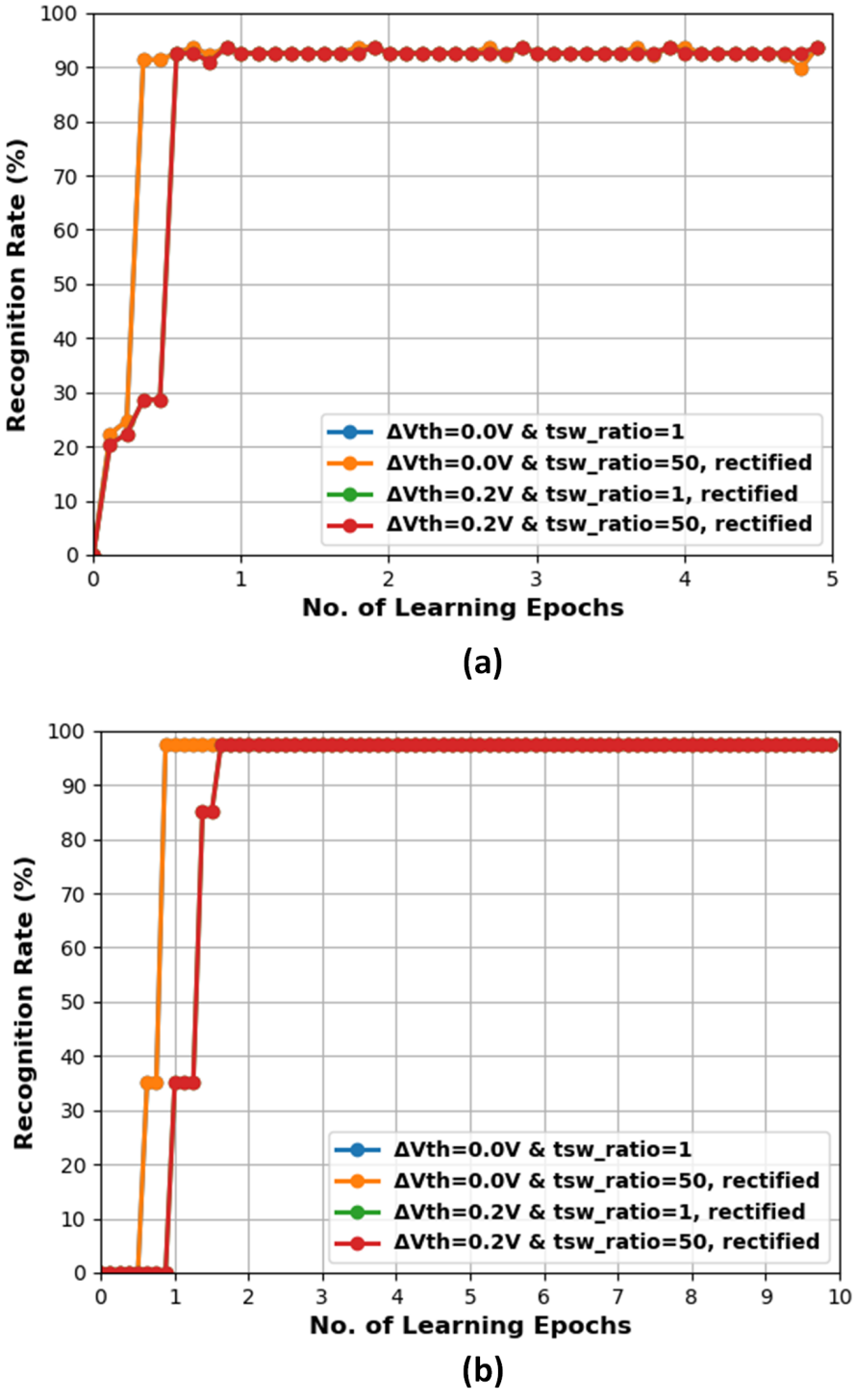

Figure 6.10: The learning in the readout layer after the asymmetry rectification methods were applied for (a) WBC dataset (b) EEG dataset.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In conclusion, this work has addressed the hardware design of memristor based neuromorphic systems with a focus on making them robust against any device level issues which are prevalent among contemporary devices. This work has proposed device-aware circuit design techniques to alleviate the adverse effects of these device level non-ideal behaviors. Through meticulous design of synapse and neuron circuits, it was demonstrated that the detrimental effects of device level issues can be mitigated. System level design was also shown with the proposed circuits to demonstrate the robustness gained by the use of the proposed techniques. The presented work can be summarized as follows:

- A bi-memristor synapse has been proposed which can implement both positive and negative weights without any additional control circuitry local to the synapse. The learning of the synapse is purely carried out due to the overlap of spikes at the either end of the synapse.

- A neuron spike shape is designed which is discrete in time and voltage such that the weight update magnitude can be carefully controlled with the spike shape. This discretization of the spike in both ways allows us to control both of the factors that contribute to the input voltage flux applied to the memristors.

- The STDP character of the bi-memristor synapse is studied through simulations. The effect of clock frequency used and device switching asymmetry on STDP behavior is studied. It is shown that by the careful choice of clock frequency, the STDP weight update magnitude can be adjusted and tuned and thereby the steepness of the STDP behavior of the synapse may be controlled. Moreover, by changing the duty cycle of the clock signals used, it is shown that the issue of switching speed asymmetry of the memristor devices may be mitigated, which would have otherwise crippled the STDP characteristic of the synapse.

- A mixed-mode neuron design is presented such that its accumulation rate can be tuned on-chip. This neuron was designed so that it can be applied to work with a wide range of memristor devices that have varying ranges for their LRS and HRS values. This neuron will thus prevent (the otherwise needed) the requirement to custom design a neuron to suite a specific memristor used.

- A crossbar system with a WTA scheme is presented for supervised STDP based learning. The proposed synapse and neuron circuits are employed in the crossbar. A detailed logic design scheme for the WTA was shown using the proposed digital encoding based neurons. A pattern recognition application was performed using the designed crossbar.

- To show the impact of device level issues at the system level, the pattern recognition application on the crossbar system was simulated along with several memristor device related issues. It is demonstrated that by virtue of using the proposed synapse and neuron in the crossbar, the system level impact of device level issues can be mitigated.

- A memristive neuromorphic circuit based liquid state machine design is presented such that the reservoir is generated using an evolutionary optimization based algorithm and the readout layer is implemented by the earlier proposed crossbar system. By using an EO based algorithm, the process of finding an optimized reservoir was streamlined whereas the use of a memristive crossbar based readout layer leads to a hardware efficient implementation of learning in this layer.

- A generalized framework for the implementation of memristor spiking neural network based reservoir computing was proposed. This layer uses binning based input encoding scheme. A general mrDANNA architecture was proposed for the reservoir implementation. This framework can help realize any given topology of a memristive neural network based liquid state machine.

- The proposed reservoir system was modeled in a high level language and classification applications were performed. The effect of memristor device switching issues on the readout layer's learning was analyzed at the application level. It is seen that they impact learning in this layer (thus impacting the classification accuracy). It is also shown that by virtue of adopting the proposed circuits and the crossbar design, this impact can be mitigated.

7.2 Future Work

As an extension of the work presented here, the following research directions can be explored for furthering this topic:

- Memristors often have limited resolution. This implies that they can only be programmed to specifc number of discrete levels between their extreme values i.e., LRS and HRS. The crossbar based supervised learning system presented here can be designed taking into consideration this fact. This can have a significant impact because this will limit the resolution of the weights that are eventually learnt by the system.

- At the crossbar level implementation of pattern recognition systems, hardware effects such as those due the parasitic resistance and capacitance of the column (interconnect) can have a detrimental effect on their learning and operation. Hence, such system level circuit integrity issues can be modeled and analyzed in combination with the circuits presented in this work.

- The technique used and the circuits developed in the presented crossbar was for supervised learning. This work can be extended to design an unsupervised learning

system. Such systems are popular and typically employ much higher number of columns to achieve a good accuracy. Hence, designing efficient and robust circuits for such applications is crucial and the techniques presented here can help achieve good robustness for such bigger systems.

- Although a single layer crossbar based pattern recognition application has been considered here, the proposed circuit techniques may be extended and applied for multi-layer neural network realizations.

Bibliography

[1] Adhikari, S. P., Yang, C., Kim, H., and Chua, L. O. (2012). Memristor bridge synapse-based neural network and its learning. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1426–1435. 2, 12

[2] Adnan, M. M., Amer, S., Sayyaparaju, S., Hasan, M. S., and Rose, G. S. (2019). A scan register based access scheme for multilevel non-volatile memristor memory. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 630–633. IEEE. 2

[3] Adnan, M. M., Sayyaparaju, S., Rose, G. S., Schuman, C. D., Ku, B. W., and Lim, S. K. (2018). A twin memristor synapse for spike timing dependent learning in neuromorphic systems. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pages 37–42. IEEE. 13

[4] Ageev, O., Blinov, Y. F., Il'in, O., Kolomiitsev, A., Konoplev, B., Rubashkina, M., Smirnov, V., and Fedotov, A. (2013). Memristor effect on bundles of vertically aligned carbon nanotubes tested by scanning tunnel microscopy. *Technical Physics*, 58(12):1831–1836. 9

[5] Alibart, F., Zamanidoost, E., and Strukov, D. B. (2013). Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature communications*, 4:2072. 2

[6] Amer, S., Hasan, M. S., Adnan, M. M., and Rose, G. S. (2018). Spice modeling of insulator metal transition: Model of the critical temperature. *IEEE Journal of the Electron Devices Society*, 7:18–25. 9

[7] Amer, S., Hasan, M. S., and Rose, G. S. (2017a). Analysis and modeling of electroforming in transition metal oxide-based memristors and its impact on crossbar array density. *IEEE Electron Device Letters*, 39(1):19–22. 41

[8] Amer, S., Rose, G. S., Beckmann, K., and Cady, N. C. (2017b). Design techniques for in-field memristor forming circuits. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1224–1227. IEEE. 9

[9] Amer, S., Sayyaparaju, S., Rose, G. S., Beckmann, K., and Cady, N. C. (2017c). A practical hafnium-oxide memristor model suitable for circuit design and simulation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE. 11

[10] Appeltant, L., Soriano, M. C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C. R., and Fischer, I. (2011). Information processing using a single dynamical node as complex system. *Nature communications*, 2(1):1–6. 66

[11] Beckmann, K., Holt, J., Manem, H., Van Nostrand, J., and Cady, N. C. (2016). Nanoscale hafnium oxide rram devices exhibit pulse dependent behavior and multi-level resistance capability. *Mrs Advances*, 1(49):3355–3360. 23

[12] Beckmann, K., Olin-Ammentorp, W., Chakma, G., Amer, S., Rose, G. S., Hobbs, C., Nostrand, J. V., Rodgers, M., and Cady, N. C. (2020). Towards synaptic behavior of nanoscale reram devices for neuromorphic computing applications. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16(2):1–18. 9

[13] Berzina, T., Erokhina, S., Camorani, P., Konovalov, O., Erokhin, V., and Fontana, M. (2009). Electrochemical control of the conductivity in an organic memristor: a time-resolved x-ray fluorescence study of ionic drift as a function of the applied voltage. *ACS Applied materials & interfaces*, 1(10):2115–2118. 9

[14] Bi, G.-q. and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472. 2

[15] Boyn, S., Grollier, J., Lecerf, G., Xu, B., Locatelli, N., Fusil, S., Girod, S., Carrétéro, C., Garcia, K., Xavier, S., et al. (2017). Learning through ferroelectric domain dynamics in solid-state synapses. *Nature communications*, 8(1):1–7. 9

[16] Bürger, J. and Teuscher, C. (2013). Variation-tolerant computing with memristive reservoirs. In *Proceedings of the 2013 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 1–6. IEEE Press. 66

[17] Campbell, K. A., Drake, K. T., and Barney Smith, E. H. (2016). Pulse shape and timing dependence on the spike-timing dependent plasticity response of ion-conducting memristors as synapses. *Frontiers in bioengineering and biotechnology*, 4:97. 62

[18] Cantley, K. D., Subramaniam, A., Stiegler, H. J., Chapman, R. A., and Vogel, E. M. (2012). Neural learning circuits utilizing nano-crystalline silicon transistors and memristors. *IEEE transactions on neural networks and learning systems*, 23(4):565–573. 9

[19] Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., and Rose, G. S. (2018). Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):125–136. 13, 15, 61, 62, 70

[20] Chakma, G., Sayyaparaju, S., Weiss, R., and Rose, G. S. (2017). A mixed-signal approach to memristive neuromorphic system design. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 547–550. IEEE. 13

[21] Chanthbouala, A., Garcia, V., Cherifi, R. O., Bouzehouane, K., Fusil, S., Moya, X., Xavier, S., Yamada, H., Deranlot, C., Mathur, N. D., et al. (2012). A ferroelectric memristor. *Nature materials*, 11(10):860–864. 9

[22] Chen, B., Lu, Y., Gao, B., Fu, Y., Zhang, F., Huang, P., Chen, Y., Liu, L., Liu, X., Kang, J., et al. (2011). Physical mechanisms of endurance degradation in tmo-rram. In *2011 International Electron Devices Meeting*, pages 12–3. IEEE. 59

[23] Chen, L., Li, C., Huang, T., Chen, Y., and Wang, X. (2014). Memristor crossbar-based unsupervised image learning. *Neural Computing and Applications*, 25(2):393–400. 2

[24] Chouard, T. and Venema, L. (2015). Machine intelligence. 1

[25] Chu, M., Kim, B., Park, S., Hwang, H., Jeon, M., Lee, B. H., and Lee, B.-G. (2015). Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron. *IEEE Transactions on Industrial Electronics*, 62(4):2410–2419. 2

[26] Chua, L. (1971). Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519. 2, 8

[27] Covi, E., Brivio, S., Serb, A., Prodromakis, T., Fanciulli, M., and Spiga, S. (2016a). Analog memristive synapse in spiking networks implementing unsupervised learning. *Frontiers in neuroscience*, 10:482. 2, 12

[28] Covi, E., Brivio, S., Serb, A., Prodromakis, T., Fanciulli, M., and Spiga, S. (2016b). Hfo2-based memristors for neuromorphic applications. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 393–396. IEEE. 2

[29] Du, C., Cai, F., Zidan, M. A., Ma, W., Lee, S. H., and Lu, W. D. (2017). Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):2204. 66

[30] Du, C., Ma, W., Chang, T., Sheridan, P., and Lu, W. D. (2015). Biorealistic implementation of synaptic functions with oxide memristors through internal ionic dynamics. *Advanced Functional Materials*, 25(27):4290–4299. 9

[31] Dua, D. and Graff, C. (2017). UCI machine learning repository. 48, 72

[32] Graf, H., Jackel, L., Howard, R., Straughn, B., Denker, J., Hubbard, W., Tennant, D., and Schwartz, D. (1986). Vlsi implementation of a neural network memory with several hundreds of neurons. In *AIP conference proceedings*, volume 151, pages 182–187. AIP. 2

[33] Hansen, M., Zahari, F., Ziegler, M., and Kohlstedt, H. (2017). Double-barrier memristive devices for unsupervised learning and pattern recognition. *Frontiers in neuroscience*, 11:91. 2, 12

[34] Hasan, R., Taha, T. M., and Yakopcic, C. (2017). On-chip training of memristor crossbar based multi-layer neural networks. *Microelectronics Journal*, 66:31–40. 13

[35] Hashem, N. and Das, S. (2012). Switching-time analysis of binary-oxide memristors via a nonlinear model. *Applied Physics Letters*, 100(26):262106. 23

[36] Hassan, A. M., Li, H. H., and Chen, Y. (2017). Hardware implementation of echo state networks using memristor double crossbar arrays. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2171–2177. IEEE. 67

[37] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W. (2011). Towards a theoretical foundation for morphological computation with compliant bodies. *Biological cybernetics*, 105(5-6):355–370. 66

[38] He, W., Huang, K., Ning, N., Ramanathan, K., Li, G., Jiang, Y., Sze, J., Shi, L., Zhao, R., and Pei, J. (2014). Enabling an integrated rate-temporal learning scheme on memristor. *Scientific reports*, 4:4755. 12

[39] Hu, M., Chen, Y., Yang, J. J., Wang, Y., and Li, H. H. (2017). A compact memristor-based dynamic synapse for spiking neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(8):1353–1366. 62

[40] Hu, P., Li, X., Lu, J., Yang, M., Lv, Q., and Li, S. (2011). Oxygen deficiency effect on resistive switching characteristics of copper oxide thin films. *Physics Letters A*, 375(18):1898–1902. 9

[41] Hu, Z., Li, Q., Li, M., Wang, Q., Zhu, Y., Liu, X., Zhao, X., Liu, Y., and Dong, S. (2013). Ferroelectric memristor based on pt/bifeo3/nb-doped srtio3 heterostructure. *Applied Physics Letters*, 102(10):102901. 9

[42] Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73. 14

[43] Jeong, D. S., Kim, K. M., Kim, S., Choi, B. J., and Hwang, C. S. (2016). Memristors for energy-efficient new computing paradigms. *Advanced Electronic Materials*, 2(9):1600090. 1

[44] Jiang, L., Lv, F.-C., Yang, R., Hu, D.-C., and Guo, X. (2018). Forming-free artificial synapses with ag point contacts at interface. *Journal of Materiomics*. 28

[45] Jin, Y. and Li, P. (2017). Performance and robustness of bio-inspired digital liquid state machines: A case study of speech recognition. *Neurocomputing*, 226:145–160. 66

[46] Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301. 2, 9, 12

[47] Kaneko, Y., Nishitani, Y., Ueda, M., and Tsujimura, A. (2013). Neural network based on a three-terminal ferroelectric memristor to enable on-chip pattern recognition. In *2013 Symposium on VLSI Technology*, pages T238–T239. IEEE. 9

[48] Kim, H., Sah, M. P., Yang, C., Roska, T., and Chua, L. O. (2012). Neural synaptic weighting with a pulse-based memristor circuit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(1):148–158. 12

[49] Kim, S., Choi, S., Lee, J., and Lu, W. D. (2014). Tuning resistive switching characteristics of tantalum oxide memristors through si doping. *ACS nano*, 8(10):10262–10269. 9

[50] Kim, S., Kim, H., Hwang, S., Kim, M.-H., Chang, Y.-F., and Park, B.-G. (2017). Analog synaptic behavior of a silicon nitride memristor. *ACS applied materials & interfaces*, 9(46):40420–40427. 9

[51] Kim, S., Lim, M., Kim, Y., Kim, H.-D., and Choi, S.-J. (2018). Impact of synaptic device variations on pattern recognition accuracy in a hardware neural network. *Scientific reports*, 8(1):2638. 12

[52] Kudithipudi, D., Saleh, Q., Merkel, C., Thesing, J., and Wysocki, B. (2016). Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing. *Frontiers in neuroscience*, 9:502. 66

[53] Kulkarni, M. S. and Teuscher, C. (2012). Memristor-based reservoir computing. In *2012 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 226–232. IEEE. 66

[54] Lecerf, G., Tomas, J., and Saïghi, S. (2013). Excitatory and inhibitory memristive synapses for spiking neural networks. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 1616–1619. IEEE. 12

[55] Lepri, S., Giacomelli, G., Politi, A., and Arecchi, F. (1994). High-dimensional chaos in delayed dynamical systems. *Physica D: Nonlinear Phenomena*, 70(3):235–249. 66

[56] Li, C., Belkin, D., Li, Y., Yan, P., Hu, M., Ge, N., Jiang, H., Montgomery, E., Lin, P., Wang, Z., et al. (2018). Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications*, 9(1):2385. 2

[57] Li, C. and Xia, Q. (2019). Three-dimensional crossbar arrays of self-rectifying si/sio 2/si memristors. In *Handbook of Memristor Networks*, pages 791–813. Springer. 9

[58] Li, Y., Zhong, Y., Xu, L., Zhang, J., Xu, X., Sun, H., and Miao, X. (2013). Ultrafast synaptic events in a chalcogenide memristor. *Scientific Reports*, 3:1619. 9

[59] Li, Y., Zhong, Y., Zhang, J., Xu, L., Wang, Q., Sun, H., Tong, H., Cheng, X., and Miao, X. (2014). Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems. *Scientific reports*, 4:4906. 9

[60] Linares-Barranco, B., Serrano-Gotarredona, T., Camuñas-Mesa, L. A., Perez-Carrasco, J. A., Zamarreño-Ramos, C., and Masquelier, T. (2011). On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in neuroscience*, 5:26. 2, 14

[61] Lu, K., Li, Y., He, W.-F., Chen, J., Zhou, Y.-X., Duan, N., Jin, M.-M., Gu, W., Xue, K.-H., Sun, H.-J., et al. (2018). Diverse spike-timing-dependent plasticity based on multilevel hfo x memristor for neuromorphic computing. *Applied Physics A*, 124(6):438. 28

[62] Manem, H., Rose, G. S., He, X., and Wang, W. (2010). Design considerations for variation tolerant multilevel cmos/nano memristor memory. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 287–292. 63

[63] Maymandi-Nejad, M. and Sachdev, M. (2003). A digitally programmable delay element: design and analysis. *IEEE transactions on very large scale integration (VLSI) systems*, 11(5):871–878. 25

[64] Medeiros-Ribeiro, G., Perner, F., Carter, R., Abdalla, H., Pickett, M. D., and Williams, R. S. (2011). Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution. *Nanotechnology*, 22(9):095702. 9

[65] Merkel, C., Saleh, Q., Donahue, C., and Kudithipudi, D. (2014). Memristive reservoir computing architecture for epileptic seizure detection. *Procedia Computer Science*, 41:249–254. 66

[66] Moradi, S. and Indiveri, G. (2014). An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE transactions on biomedical circuits and systems*, 8(1):98–107. 2

[67] Morishita, T., Tamura, Y., Otsuki, T., and KANO, G. (1992). A bicmos analog neural network with dynamically updated weights. *IEICE Transactions on Electronics*, 75(3):297–302. 2

[68] Najem, J. S., Taylor, G. J., Armendarez, N., Weiss, R. J., Hasan, M. S., Rose, G. S., Schuman, C. D., Belianinov, A., Sarles, S. A., and Collier, C. P. (2019). Assembly and characterization of biomolecular memristors consisting of ion channel-doped lipid membranes. *JoVE (Journal of Visualized Experiments)*, (145):e58998. 9

[69] Najem, J. S., Taylor, G. J., Weiss, R. J., Hasan, M. S., Rose, G., Schuman, C. D., Belianinov, A., Collier, C. P., and Sarles, S. A. (2018). Memristive ion channel-doped biomembranes as synaptic mimics. *ACS nano*, 12(5):4702–4711. 9

[70] Nakane, R., Tanaka, G., and Hirose, A. (2018). Reservoir computing with spin waves excited in a garnet film. *IEEE Access*, 6:4462–4469. 66

[71] Nishitani, Y., Kaneko, Y., and Ueda, M. (2015). Supervised learning using spike-timing-dependent plasticity of memristive synapses. *IEEE transactions on neural networks and learning systems*, 26(12):2999–3008. 2

[72] Oblea, A. S., Timilsina, A., Moore, D., and Campbell, K. A. (2010). Silver chalcogenide based memristor devices. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–3. IEEE. 9

[73] Panchula, A. (2007). Oscillating-field assisted spin torque switching of a magnetic tunnel junction memory element. US Patent 7,224,601. 9

[74] Park, S., Chu, M., Kim, J., Noh, J., Jeon, M., Lee, B. H., Hwang, H., Lee, B., and Lee, B.-g. (2015). Electronic system with memristive synapses for pattern recognition. *Scientific reports*, 5:10123. 2

[75] Payvand, M., Rofeh, J., Sodhi, A., and Theogarajan, L. (2014). A cmos-memristive self-learning neural network for pattern classification applications. In *Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 92–97. ACM. 2, 12

[76] Pedretti, G., Milo, V., Ambrogio, S., Carboni, R., Bianchi, S., Calderoni, A., Ramaswamy, N., Spinelli, A., and Ielmini, D. (2017). Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity. *Scientific reports*, 7(1):5288. 1, 2, 12

[77] Pi, S., Li, C., Jiang, H., Xia, W., Xin, H., Yang, J. J., and Xia, Q. (2019). Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nature nanotechnology*, 14(1):35–39. 9

[78] Polepalli, A., Soures, N., and Kudithipudi, D. (2016). Reconfigurable digital design of a liquid state machine for spatio-temporal data. In *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication*, page 15. ACM. 66

[79] Pouyan, P., Amat, E., and Rubio, A. (2015). Statistical lifetime analysis of memristive crossbar matrix. In *2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6. IEEE. 59

[80] Prezioso, M., Bayat, F. M., Hoskins, B., Likharev, K., and Strukov, D. (2016). Self-adaptive spike-time-dependent plasticity of metal-oxide memristors. *Scientific reports*, 6(1):1–6. 9

[81] Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61. 2

[82] Querlioz, D., Zhao, W., Dollfus, P., Klein, J.-O., Bichler, O., and Gamrat, C. (2012). Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches. In *2012 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 203–210. IEEE. 2

[83] Radoi, A., Dragoman, M., and Dragoman, D. (2011). Memristor device based on carbon nanotubes decorated with gold nanoislands. *Applied Physics Letters*, 99(9):093102. 9

[84] Rahaman, S. Z., Lin, Y.-D., Lee, H.-Y., Chen, Y.-S., Chen, P.-S., Chen, W.-S., Hsu, C.-H., Tsai, K.-H., Tsai, M.-J., and Wang, P.-H. (2017). The role of ti buffer layer thickness on the resistive switching properties of hafnium oxide-based resistive switching memories. *Langmuir*, 33(19):4654–4665. 9

[85] Ramakrishnan, S., Hasler, P. E., and Gordon, C. (2011). Floating gate synapses with spike-time-dependent plasticity. *IEEE Transactions on Biomedical Circuits and Systems*, 5(3):244–252. 2

[86] Reynolds, J. J. M., Plank, J. S., and Schuman, C. D. (2019). Intelligent reservoir generation for liquid state machines using evolutionary optimization. In *IJCNN: The International Joint Conference on Neural Networks*, Budapest. 67

[87] Saxena, V., Wu, X., Srivastava, I., and Zhu, K. (2018). Towards neuromorphic learning machines using emerging memory devices with brain-like energy efficiency. *Journal of Low Power Electronics and Applications*, 8(4):34. 13, 50, 60

[88] Sayyaparaju, S., Adnan, M. M., Amer, S., and Rose, G. S. (2020). Device-aware circuit design for robust memristive neuromorphic systems with stdp-based learning. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16(3):1–25. 15

[89] Sayyaparaju, S., Amer, S., and Rose, G. S. (2018a). A bi-memristor synapse with spike-timing-dependent plasticity for on-chip learning in memristive neuromorphic systems. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 69–74. IEEE. 15

[90] Sayyaparaju, S., Chakma, G., Amer, S., and Rose, G. S. (2017). Circuit techniques for online learning of memristive synapses in cmos-memristor neuromorphic systems. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 479–482. 13

[91] Sayyaparaju, S., Weiss, R., and Rose, G. S. (2018b). A mixed-mode neuron with on-chip tunability for generic use in memristive neuromorphic systems. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 441–446. IEEE. 35, 61, 103

[92] Schrauwen, B., D'Haene, M., Verstraeten, D., and Van Campenhout, J. (2008). Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural networks*, 21(2-3):511–523. 66

[93] Schrauwen, B., Verstraeten, D., and Van Campenhout, J. (2007). An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pages 471–482. 64

[94] Schuman, C. D., Plank, J. S., Bruer, G., and Anantharaj, J. (2019). Non-traditional input encoding schemes for spiking neuromorphic systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE. 68

[95] Schürmann, F., Meier, K., and Schemmel, J. (2005). Edge of chaos computation in mixed-mode vlsi-a hard liquid. In *Advances in neural information processing systems*, pages 1201–1208. 66

[96] Seo, J.-s., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., Rajendran, B., Tierno, J. A., Chang, L., Modha, D. S., et al. (2011). A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE. 2

[97] Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013). Stdp and stdp variations with memristors for spiking neuromorphic learning systems. *Frontiers in neuroscience*, 7:2. 12

[98] Sheri, A. M., Hwang, H., Jeon, M., and Lee, B.-g. (2014). Neuromorphic character recognition system with two pcmo memristors as a synapse. *IEEE Transactions on Industrial Electronics*, 61(6):2933–2941. 2, 13

[99] Sheridan, P., Ma, W., and Lu, W. (2014). Pattern recognition with memristor networks. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1078–1081. IEEE. 2

[100] Snider, G. S. (2008). Spike-timing-dependent learning in memristive nanodevices. In *Proceedings of the 2008 IEEE International Symposium on Nanoscale Architectures*, pages 85–92. IEEE Computer Society. 2, 12

[101] Soures, N., Hays, L., and Kudithipudi, D. (2017a). Robustness of a memristor based liquid state machine. In *2017 international joint conference on neural networks (ijcnn)*, pages 2414–2420. IEEE. 67

[102] Soures, N., Merkel, C., Kudithipudi, D., Thiem, C., and McDonald, N. (2017b). Reservoir computing in embedded systems: Three variants of the reservoir algorithm. *IEEE Consumer Electronics Magazine*, 6(3):67–73. 66

[103] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *nature*, 453(7191):80. 2, 9

[104] Sun, W., Gao, B., Chi, M., Xia, Q., Yang, J. J., Qian, H., and Wu, H. (2019). Understanding memristive switching via in situ characterization and device modeling. *Nature communications*, 10(1):1–13. 9

[105] Sun, Y., Xu, H., Wang, C., Song, B., Liu, H., Liu, Q., Liu, S., and Li, Q. (2018). A ti/alo x/tao x/pt analog synapse for memristive neural network. *IEEE Electron Device Letters*, 39(9):1298–1301. 21, 28

[106] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*. 66

[107] Tang, Y., Nyengaard, J. R., De Groot, D. M., and Gundersen, H. J. G. (2001). Total regional and global number of synapses in the human brain neocortex. *Synapse*, 41(3):258–273. 2

[108] Torrejon, J., Riou, M., Araujo, F. A., Tsunegi, S., Khalsa, G., Querlioz, D., Bortolotti, P., Cros, V., Yakushiji, K., Fukushima, A., et al. (2017). Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547(7664):428–431. 66

[109] Urbain, G., Degrave, J., Carette, B., Dambre, J., and Wyffels, F. (2017). Morphological properties of mass–spring networks for optimal locomotion learning. *Frontiers in neurorobotics*, 11:16. 66

[110] Vandoorne, K., Dambre, J., Verstraeten, D., Schrauwen, B., and Bienstman, P. (2011). Parallel reservoir computing using optical amplifiers. *IEEE transactions on neural networks*, 22(9):1469–1481. 66

[111] Vandoorne, K., Dierckx, W., Schrauwen, B., Verstraeten, D., Baets, R., Bienstman, P., and Van Campenhout, J. (2008). Toward optical signal processing using photonic reservoir computing. *Optics express*, 16(15):11182–11192. 66

[112] Wang, Q., Jin, Y., and Li, P. (2015). General-purpose lsm learning processor architecture and theoretically guided design space exploration. In *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE. 66

[113] Wang, Q., Li, Y., and Li, P. (2016). Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing. In *2016*

IEEE International Symposium on Circuits and Systems (ISCAS), pages 361–364. IEEE. 66

[114] Wen, S., Hu, R., Yang, Y., Huang, T., Zeng, Z., and Song, Y.-D. (2018). Memristor-based echo state network with online least mean square. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. 67

[115] Wu, X., Saxena, V., Zhu, K., and Balagopal, S. (2015). A cmos spiking neuron for brain-inspired neural networks with resistive synapses andin situlearning. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(11):1088–1092. 62

[116] Yang, J. J., Zhang, M.-X., Strachan, J. P., Miao, F., Pickett, M. D., Kelley, R. D., Medeiros-Ribeiro, G., and Williams, R. S. (2010). High switching endurance in tao x memristive devices. *Applied Physics Letters*, 97(23):232102. 9, 59

[117] Yao, P., Wu, H., Gao, B., Eryilmaz, S. B., Huang, X., Zhang, W., Zhang, Q., Deng, N., Shi, L., Wong, H.-S. P., et al. (2017). Face classification using electronic synapses. *Nature communications*, 8:15199. 2

[118] Yi, Y., Liao, Y., Wang, B., Fu, X., Shen, F., Hou, H., and Liu, L. (2016). Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors. *Microprocessors and Microsystems*, 46:175–183. 66

[119] Yilmaz, O. (2015). Symbolic computation using cellular automata-based hyperdimensional computing. *Neural computation*, 27(12):2661–2692. 66

[120] Zahari, F., Hansen, M., Mussenbrock, T., Ziegler, M., and Kohlstedt, H. (2015). Pattern recognition with tiox-based memristive devices. *AIMS Mater. Sci*, 2:203–216. 2, 12

[121] Zhang, Y., Li, P., Jin, Y., and Choe, Y. (2015). A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE transactions on neural networks and learning systems*, 26(11):2635–2649. 66

[122] Zhou, E., Fang, L., and Yang, B. (2018). Memristive spiking neural networks trained with unsupervised stdp. *Electronics*, 7(12):396. 2

[123] Zhou, Y. and Ramanathan, S. (2015). Mott memory and neuromorphic devices. *Proceedings of the IEEE*, 103(8):1289–1310. 9

[124] Zidan, M. A., Fahmy, H. A. H., Hussain, M. M., and Salama, K. N. (2013). Memristor-based memory: The sneak paths problem and solutions. *Microelectronics Journal*, 44(2):176–183. 63

Appendices

A Testing Methodology for the Test Structures

Several test structures for the mixed-mode neuron were presented in Section 4.3. In this section, further details of the test structures are presented along with their connections to the probe pads, pin-outs and the recommended testing strategies for each of these structures. In the following sub-sections, for each of the four test structures, a pin out diagram is first shown. Then, the types and purpose of each pin is delineated in a table. Later, a preferred testing strategy for them is presented in a separate table. Lastly, test results are shown where available and simulations are shown for the rest of them.

A.1 Test Setup

A probe station has been used here for probing (contacting) the probe pads to which the test structures are connected. Fig. 1 shows the microscopic view of contacting and probing these test structures. The layout of the chip containing the test structures is shown in Fig. 2. Looking at the chip with this view, the test structures are arranged in a grid fashion at the top right corner. In the subsequent sections, the location of a particular test structure on this chip is indicated using its row and column position, with the top row being the first one and the column to the left being the first column.

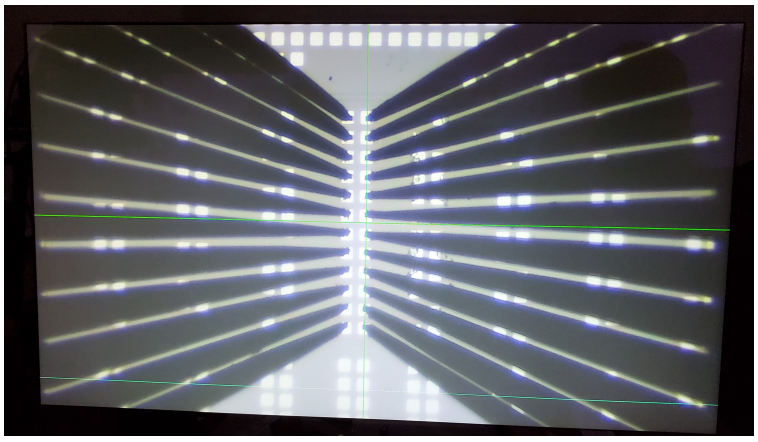

Figure 1: The view through the microscope during probing of the test structures.

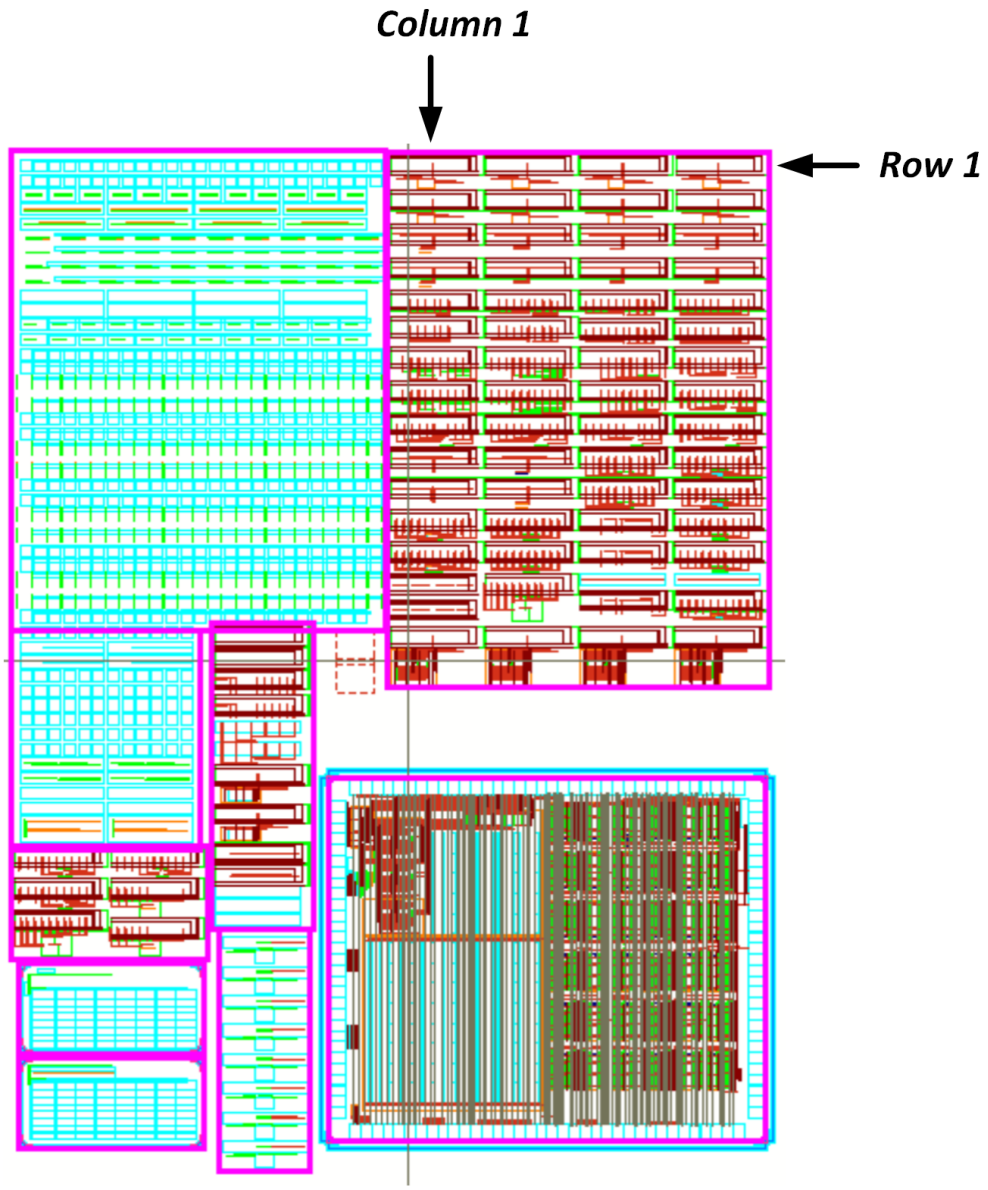

Figure 2: The layout of the chip and the naming convention used here for rows and columns of the grid for locating test structures.

A.2 Analog Block Test Structure

The pin out for this test structure when the circuit is connected to the probe pads is shown in Fig. 3. It may be noted that here, a 12x2 probe pad has been used and the pads corresponding to pins for the circuit have been labeled in this figure. Pads that have been left unnamed are floating and do not have any connection. This test structure is located at column 1 and rows 5,6.

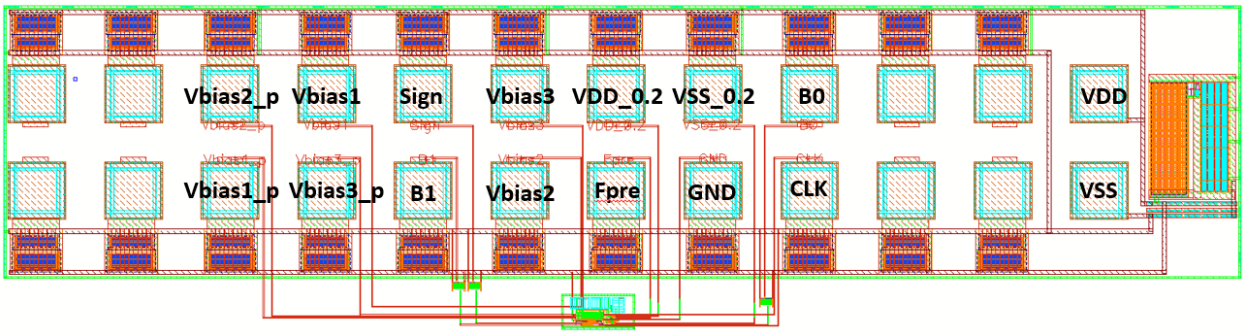

Figure 3: Pin out for the analog block test structure connected to the probe pads.

Table 1 shows the details of the pins for this test structure. It explains the type (analog/digital) and direction (input/output) for each pin and also explains the purpose and uses of them. It may be noted that the numbering of pads here is such that the VDD pad in Fig. 3 is 1 and the numbers increase to the left and wrap around such that the pad VSS is 24. Also, in this table, blank columns denote the floating probe pads.

In Table 2, the plan for the test of this structure is described. It may be seen here that the threshold voltages Vbias1-3 are the key to observe 'tuning' of encoding by this analog block. By changing these reference voltages inputs, the number of total discharged nodes in this block is varied and hence the encoded bit value will change. The threshold voltages Vbias1-3_p have an influence when there is a negative input synaptic weight, which is not the case here. Hence, it is suggested to bias them all with 0V (VSS) for simplicity.

Table 1: The pin description for the analog block test structure.

Pad Number	Pin Name	Pin type	Pin Description
1	VDD	Digital Input	Power Rail
2	—	—	—
3	—	—	—
4	B0	Digital Output	LSB of encoded bit
5	VSS_0.2	Analog DC Input	Power rail for synapse input
6	VDD_0.2	Analog DC Input	Power rail for synapse input
7	Vbias3	Analog DC Input	Reference Voltage
8	Sign	Digital Output	Sign bit of endoded value
9	Vbias1	Analog DC Input	Reference Voltage
10	Vbias2_p	Analog DC Input	Reference Voltage
11	—	—	—
12	—	—	—
13	—	—	—
14	—	—	—
15	Vbias1_p	Analog DC Input	Reference Volatge
16	Vbias3_p	Analog DC Input	Reference Voltage
17	B1	Digital Output	MSB of the encoded value
18	Vbias2	Analog DC Input	Reference Voltage
19	Fpre	Digital Input	Input Control
20	GND	Analog DC Input	Mid-Rail Voltage
21	CLK	Digital Pulse	Clock Signal
22	—	—	—
23	—	—	—
24	VSS	Digital Input	Power Rail

Table 2: The suggested test plan for the analog block test structure.

Pad Number	Pin Name	Pin Bias
1	VDD	1.2V
2	—	—
3	—	—
4	B0	Output
5	VSS_0.2	0V
6	VDD_0.2	1.2V
7	Vbias3	Apply either VDD or VSS
8	Sign	Output
9	Vbias1	Apply either VDD or VSS
10	Vbias2_p	0V
11	—	—
12	—	—
13	—	—
14	—	—
15	Vbias1_p	0V
16	Vbias3_p	0V
17	B1	Output
18	Vbias2	Apply either VDD or VSS
19	Fpre	1.2V
20	GND	0.6V
21	CLK	Clock of any frequency with quick edges
22	—	—
23	—	—
24	VSS	0V

Testing Results for the analog block test structure

The tests for this analog block test structure were performed using the fabricated chip. The chip was put in a probe station and was probed. The necessary voltage biases were applied to the test structure's inputs and the outputs were recorded on an oscilloscope. The following figures show the outputs observed for each test condition. In each such condition, the input synaptic weight is fixed in the design by the resistive synapse. The reference voltages are changed such that number of fully discharged evaluation nodes changes. Since these evaluation nodes are discharged by an NMOS device, whose source terminal is driven by the reference voltages, applying a high voltage reduces the likeliness of it switching ON. This implies that by applying high voltage such as VDD does not switching ON this device and hence the corresponding evaluation node will not be discharged. Whereas applying a low voltage at the source such as VSS will switch it ON leading to the discharge of the evaluation node. Hence, by changing the combination of the three reference voltages of this circuit, the number of fully discharged evaluation nodes is controlled and hence the 'tuning' of the encoded value is achieved and demonstrated here. Note that in this case as more evaluation nodes discharge, a higher value for the encoded value is obtained, which is in contrast to the design described in Chapter 4. This is because of the fundamental difference in the current to voltage transformation techniques adopted in the design presented therein versus the one used in this circuit on the chip as described in [91]. However, there is a conceptual similarity in the way encoding is performed and its tunability and hence these test results serve as a good proof of concept for the proposed design.

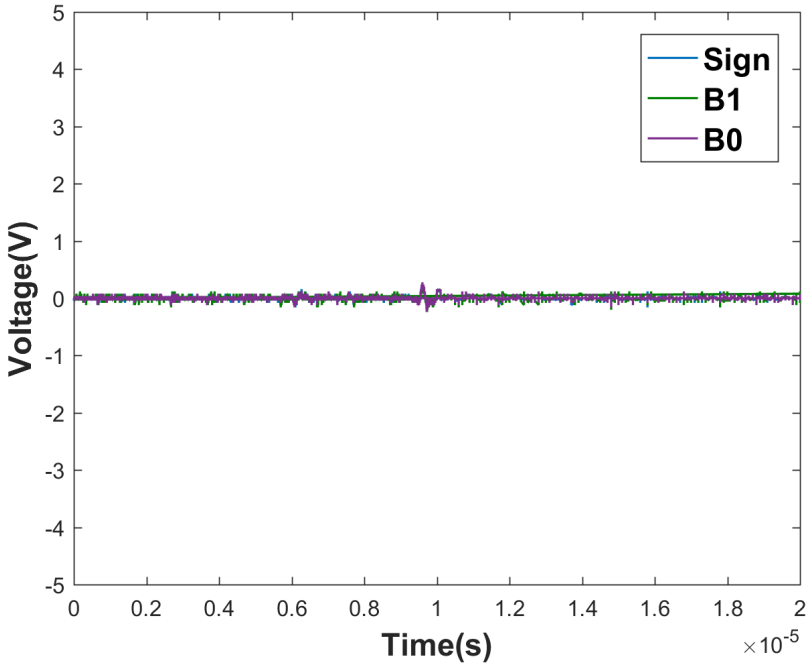

Figure 4: Encoded value of 000 for Vbias1-3 = 1.2V. None of the nodes discharge.

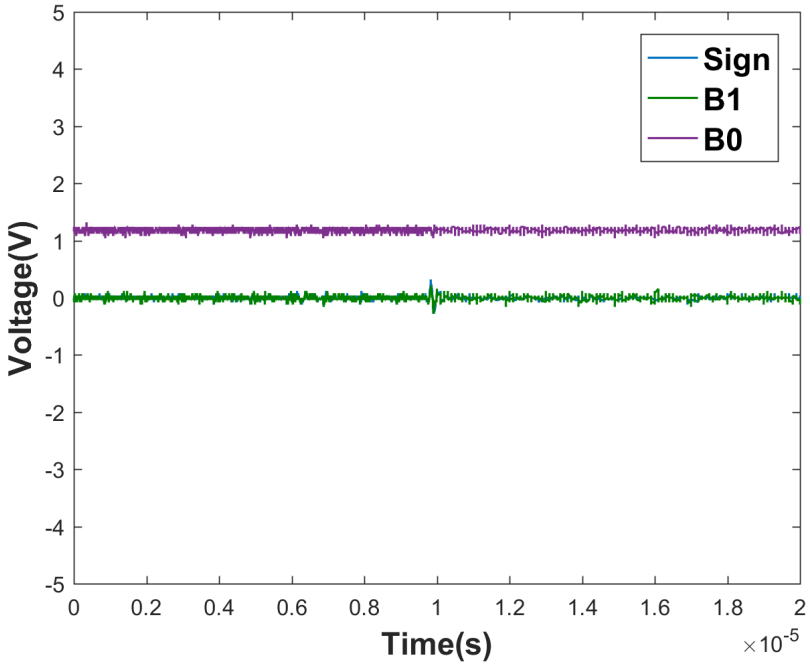

Figure 5: Encoded value of 001 for Vbias1 = 0V and Vbias2-3=1.2V. One of the nodes discharges.

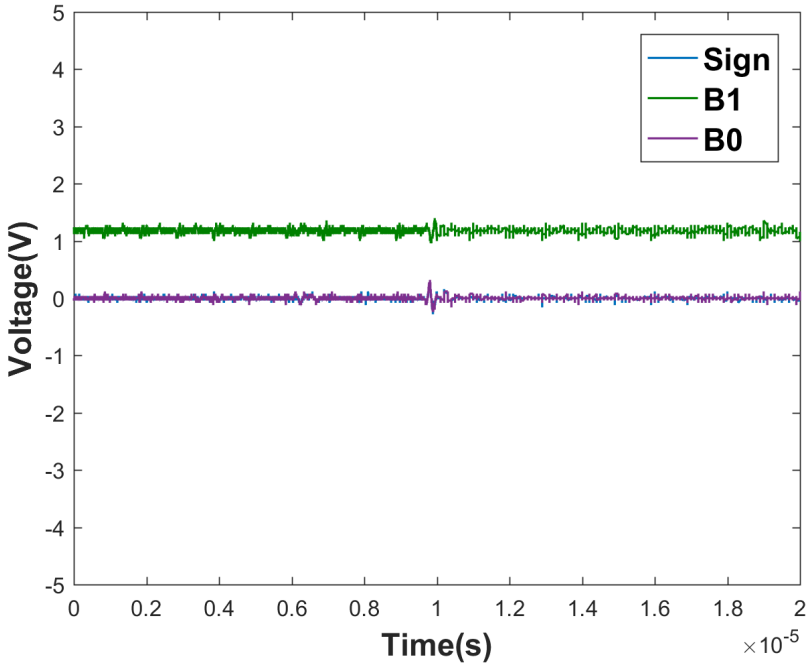

Figure 6: Encoded value of 010 for Vbias1-2 = 0V and Vbias3=1.2V. Two of the nodes discharge.

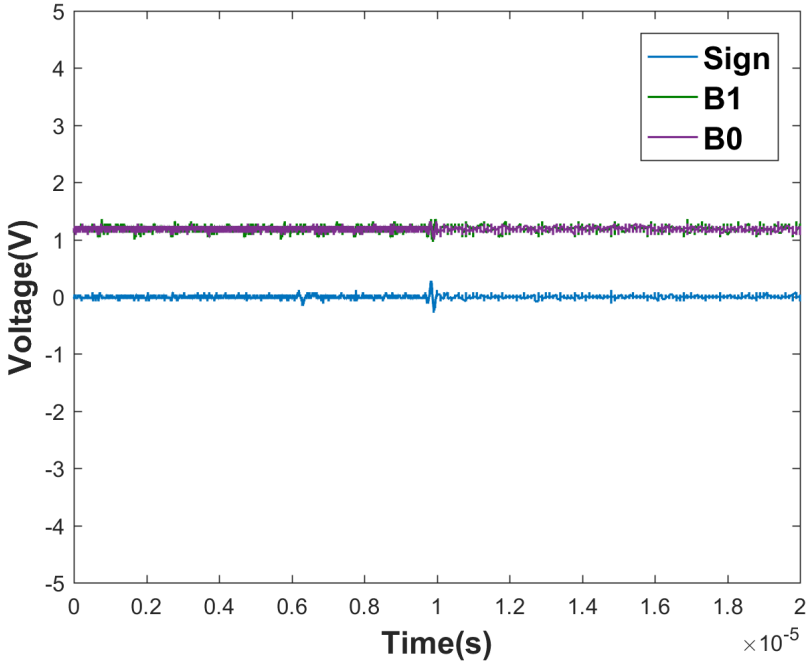

Figure 7: Encoded value of 011 for Vbias1-3 = 0V. All of the nodes discharge.

A.3 Mixed-Mode Neuron Test Structure with Single Resistive Synapse

The pin out for this test structure is shown in Fig. 8. The description of these pins is given by Table 3 and the test plan for this test structure is given by Table 4. This test structure is located at column 2 and rows 5,6. It may be noted that for this test structure, the output will be the neuron's spike (Fre). Also, the digital accumulation in the neuron (D2-D0) was taken-out on output pins. Fig. 9 shows a test result for this test structure. In this case, the reference voltages applied are Vbias1=0V and Vbias2-3=1.2V, thus leading to an accumulation of '001' per each clock cycle. This leads to the neuron accumulating this value until it reaches the threshold. This can be seen in the waveforms for D1 and D0, which are the magnitude bits. Their values cycle between 00, 01 and 10. Upon reaching 11, they reach the maximum accumulation (which is also the threshold applied here) and hence at that clock cycle, a spike condition is TRUE and therefore D1 and D0 are RESET to 00 again. It may be noted that D2 always stays at 0 because it is the sign bit and since the input is positive weight, it is always 0. For further testing of this circuit, the reference voltages combination was changed to Vbias1-2=0V and Vbias3=1.2V, thus leading to an accumulation of '010'. This can be seen in Fig. 10 wherein D1 and D0 cycle between 00 and 10 because 10+10 would lead to an overflow (spike) condition and hence their values cycle between 00 (RESET) and 10 (accumulation). These results indicate that the accumulation in the neuron (including the digital part of it) is working good and as expected.

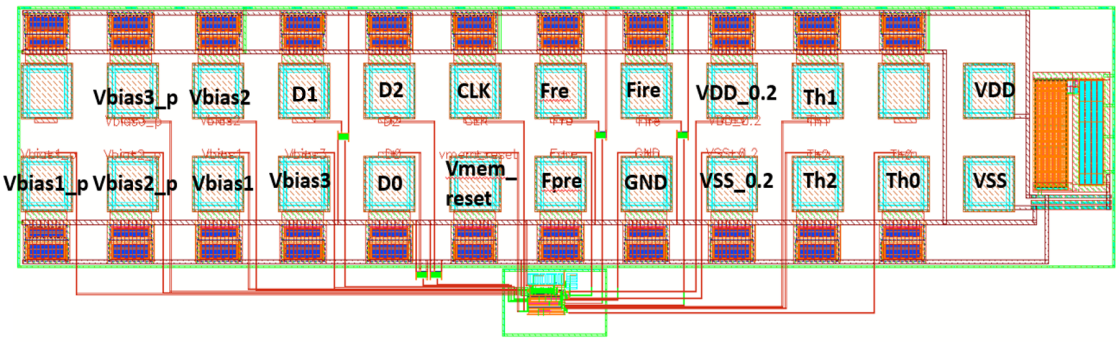

Figure 8: Pin out for the mixed-mode neuron test structure with one resistive synapse, connected to the probe pads.

Table 3: The pin description for the mixed-mode neuron test structure with one resistive synapse.

Pad Number	Pin Name	Pin type	Pin Description
1	VDD	Digital Input	Power Rail
2	—	—	—
3	Th1	Digital Input	Digital Threshold Bit
4	VDD_0.2	Analog DC Input	Power rail for synapse input
5	Fire	Digital Output	Delayed Neuron Spike
6	Fre	Digital Output	Neuron Spike
7	CLK	Digital Pulse	Clock Signal
8	D2	Digital Output	Sign bit of accumulation
9	D1	Digital Output	MSB of accumulation
10	Vbias2	Analog DC Input	Reference Voltage
11	Vbias3_p	Analog DC Input	Reference Voltage
12	—	—	—
13	Vbias1_p	Analog DC Input	Reference Volatge
14	Vbias2_p	Analog DC Input	Reference Volatge
15	Vbias1	Analog DC Input	Reference Volatge
16	Vbias3	Analog DC Input	Reference Voltage
17	D0	Digital Output	LSB of accumulation
18	vmem_reset	Digital Pulse	Registers' RESET
19	Fpre	Digital Input	Input Control
20	GND	Analog DC Input	Mid-Rail Voltage
21	VSS_0.2	Analog DC Input	Power rail for synapse input
22	Th2	Digital Input	Digital Threshold Bit
23	Th0	Digital Input	Digital Threshold Bit
24	VSS	Digital Input	Power Rail

Table 4: The suggested test plan for the mixed-mode neuron test structure with one resistive synapse.

Pad Number	Pin Name	Pin Bias
1	VDD	1.2V
2	—	—
3	Th1	1.2V
4	VDD_0.2	1.2V
5	Fire	Output
6	Fre	Output
7	CLK	Clock of any frequency with quick edges
8	D2	Output
9	D1	Output
10	Vbias2	apply either VDD or VSS
11	Vbias3_p	0V
12	—	—
13	Vbias1_p	0V
14	Vbias2_p	0V
15	Vbias1	apply either VDD or VSS
16	Vbias3	apply either VDD or VSS
17	D0	Output
18	vmem_reset	Apply a one time RESET pulse at startup
19	Fpre	Digital Pulse (half frequency of CLK)
20	GND	0.6V
21	VSS_0.2	0V
22	Th2	0V
23	Th0	1.2V
24	VSS	0V

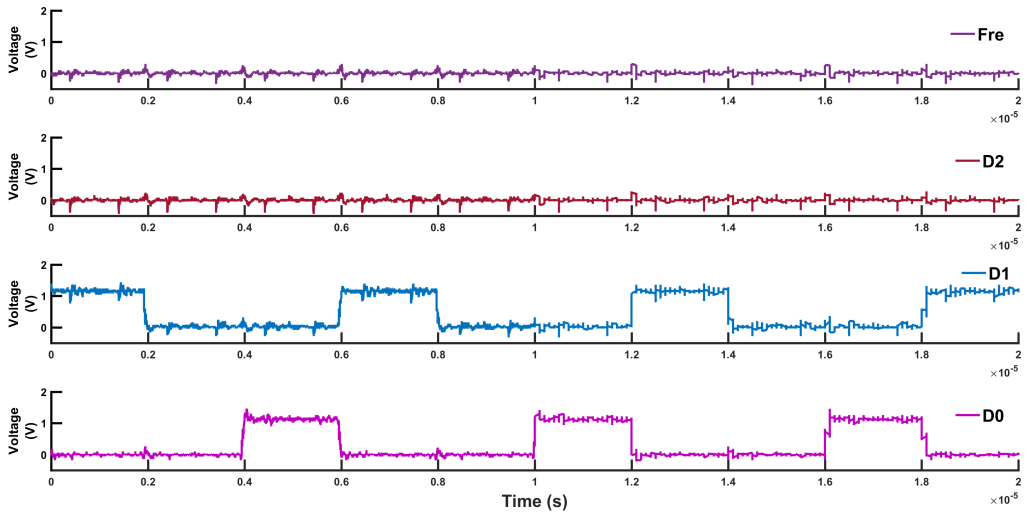

Figure 9: Test result for the neuron test structure with one resistive synapse. The reference voltages used are Vbias1=0V and Vbias2-3=1.2V. Hence, D1 and D0 bits cycle between 00, 01 and 10 values. 11 is not seen because it is the maximum accumulation, where a spike is supposed to occur and hence D1 and D0 are RESET to 0. Note that D2 is always 0 because it is the sign bit here.

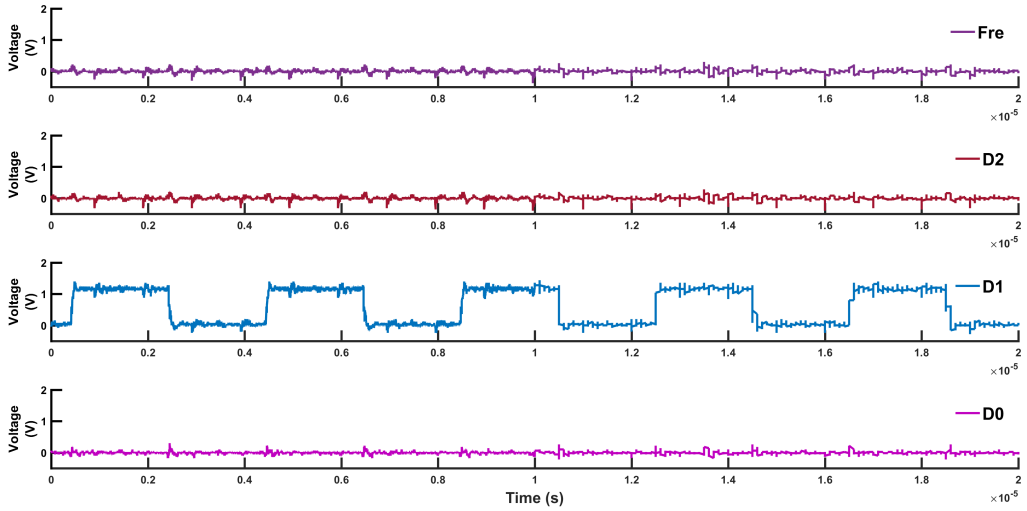

Figure 10: Test result for the neuron test structure with one resistive synapse, where the reference voltages used are Vbias1-2=0V and Vbias3=1.2V. Here, an accumulation of '10' can be observed and hence the D1-D0 outputs cycle between 00 and 10. Note that D2 is always 0 because it is the sign bit here.

It may be noted from the above test results (Fig. 9 and Fig. 10) that although the internal digital accumulation in the neuron was working correctly and as expected, the neuron spike output 'Fre' was not seen as expected. The possible reason here is a timing violation at the input of the D-flip flop controlling this signal. Inside the digital half of the neuron, when the spike condition is TRUE, the spike trigger signal is input to a D-flip flop such that its output is the output spike ('Fre') and would hold for an entire clock period. However, due to the slow clock rise and fall times experienced in the test setup used here (as seen in Fig. 11), the flip flop is unable to capture its input. Fig. 12 shows the simulation for this flip flop for the condition when the clock transition periods are slow ($\geq$1ns). It may be seen here that there is a clear timing violation (D-input transition occurring ahead of clock transition). Fig. 13 shows that this can be mitigated by using a test setup with a faster clock transition ($\leq$1ns).

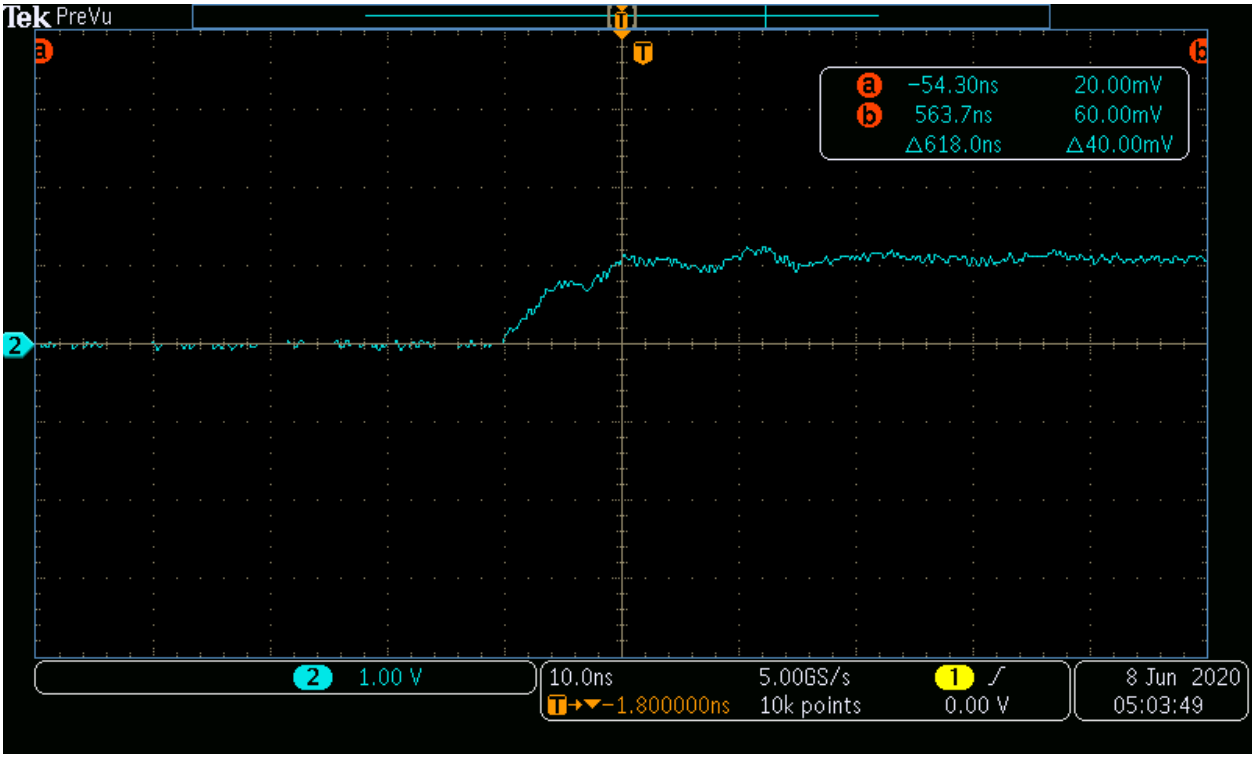

Figure 11: Zoomed-in screenshot from the oscilloscope showing the rise time of the clock signal being $\approx$ 10ns.

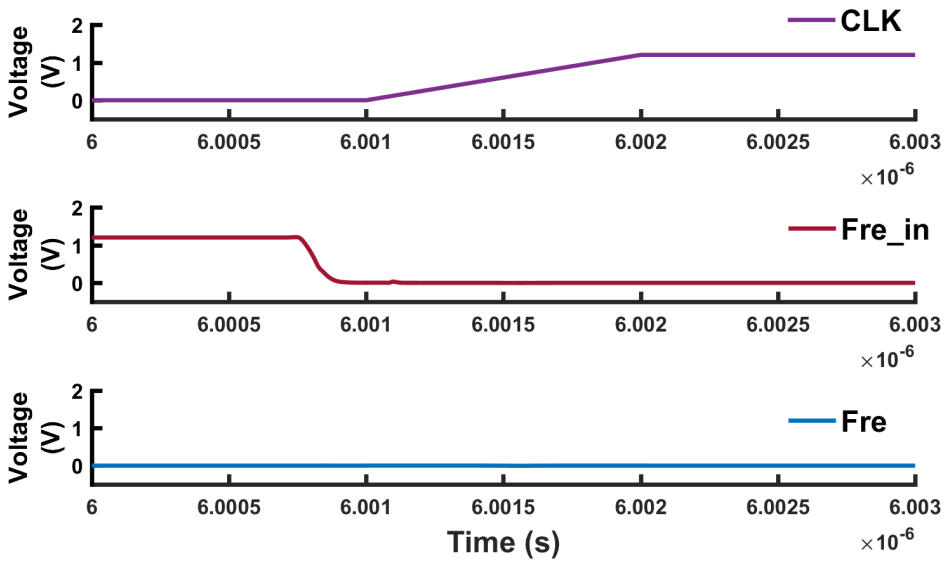

Figure 12: Simulation showing the input to the D-flip flop controlling the 'Fre' (spike) output of the neuron. It may be seen that due to the slow rise time ($\geq$1ns) of the clock signal to the D-flip flop, there is a hold time violation at its input.

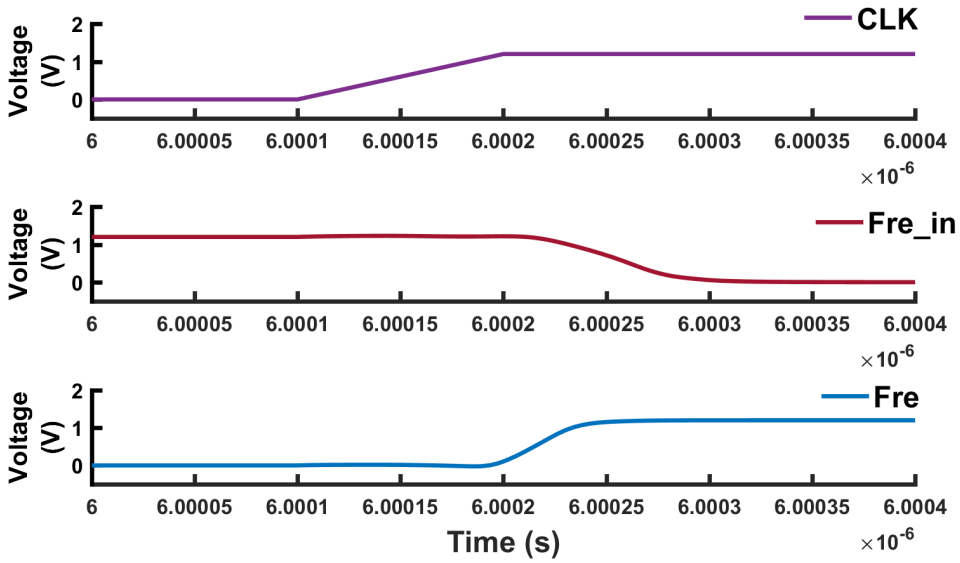

Figure 13: Simulation showing the input to the D-flip flop for the case of having a faster clock input rise/fall time. In this case a sub nanosecond (0.1ns) rise/fall was used and it may be seen that the hold time for the flip flop is satisfied here.

A.4 Mixed-Mode Neuron Test Structure with two Resistive Input Synapses

The pin out for this test structure is shown in Fig. 14. The description of these pins is given by Table 5 and the test plan for this test structure is given by Table 6. This test structure is located at column 3 and rows 5,6. As with the earlier test structure, the output here will be in terms of the neuron's spike (Fre). Also, the digital accumulation in the neuron (D2-D0) was taken-out in this test structure also, like earlier, on output pins so that it may also be monitored. Fig. 15 shows a test result for this test structure. In this case, the reference voltages applied are Vbias1-2=0V and Vbias3=1.2V, thus leading to an accumulation of '010' per each clock cycle. This can be seen in Fig. 15 where the signals D1-D0 cycle between the values 00 and 10. D2 was always 0 because the input synaptic weight is designed to be a fixed positive weight. As with the earlier test structure, the spike output 'Fre' here is not seen here possibly due to the same timing issue as explained earlier. This is expected to be rectified provided a clock with faster transition times is used.

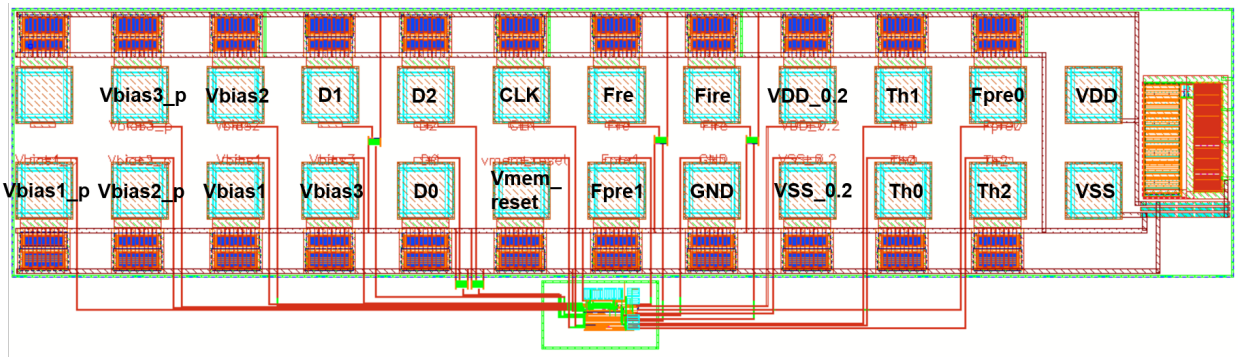

Figure 14: Pin out for the mixed-mode neuron test structure with two resistive synapses, connected to the probe pads.

Table 5: The pin description for the mixed-mode neuron test structure with two resistive synapses.

Pad Number	Pin Name	Pin type	Pin Description
1	VDD	Digital Input	Power Rail
2	Fpre0	Digital Input	Input Control
3	Th1	Digital Input	Digital Threshold Bit
4	VDD_0.2	Analog DC Input	Power rail for synapse input
5	Fire	Digital Output	Delayed Neuron Spike
6	Fre	Digital Output	Neuron Spike
7	CLK	Digital Pulse	Clock Signal
8	D2	Digital Output	Sign bit of accumulation
9	D1	Digital Output	MSB of accumulation
10	Vbias2	Analog DC Input	Reference Voltage
11	Vbias3_p	Analog DC Input	Reference Voltage
12	—	—	—
13	Vbias1_p	Analog DC Input	Reference Volatge
14	Vbias2_p	Analog DC Input	Reference Volatge
15	Vbias1	Analog DC Input	Reference Volatge
16	Vbias3	Analog DC Input	Reference Voltage
17	D0	Digital Output	LSB of accumulation
18	vmem_reset	Digital Pulse	Registers' RESET
19	Fpre1	Digital Input	Input Control
20	GND	Analog DC Input	Mid-Rail Voltage
21	VSS_0.2	Analog DC Input	Power rail for synapse input
22	Th0	Digital Input	Digital Threshold Bit
23	Th2	Digital Input	Digital Threshold Bit
24	VSS	Digital Input	Power Rail

Table 6: The suggested test plan for the mixed-mode neuron test structure with two resistive synapses.

Pad Number	Pin Name	Pin Bias
1	VDD	1.2V
2	Fpre0	Digital Pulse (half frequency of CLK)
3	Th1	1.2V
4	VDD_0.2	1.2V
5	Fire	Output
6	Fre	Output
7	CLK	Clock of any frequency with quick edges
8	D2	Output
9	D1	Output
10	Vbias2	apply either a VDD or VSS
11	Vbias3_p	0V
12	—	—
13	Vbias1_p	0V
14	Vbias2_p	0V
15	Vbias1	apply either a VDD or VSS
16	Vbias3	apply either a VDD or VSS
17	D0	Output
18	vmem_reset	Apply a one time RESET pulse at startup
19	Fpre1	Digital Pulse (half frequency of CLK)
20	GND	0.6V
21	VSS_0.2	0V
22	Th0	1.2V
23	Th2	0V
24	VSS	0V

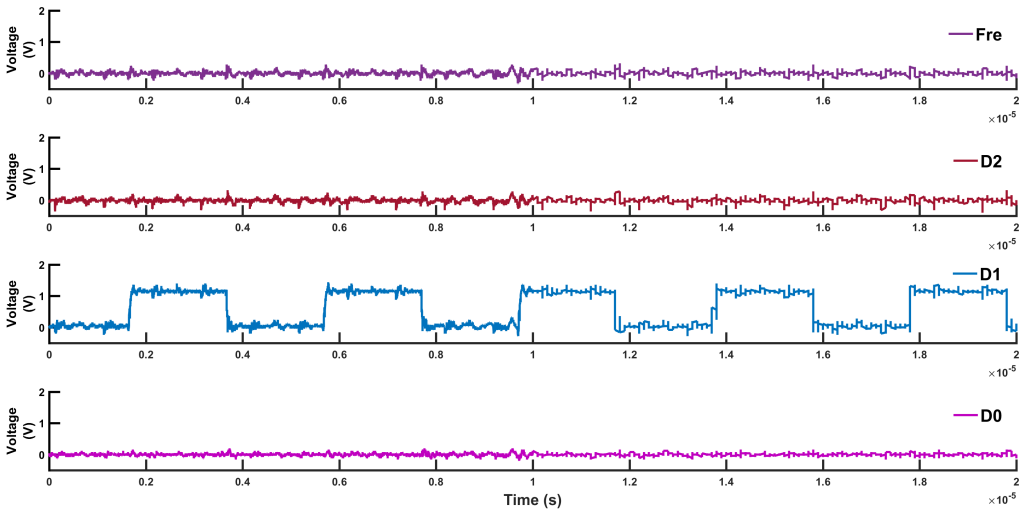

Figure 15: Test result for the neuron test structure with two resistive synapses, where the reference voltages used are Vbias1-2=0V and Vbias3=1.2V. Here, an accumulation of '10' can be observed and hence the D1-D0 outputs cycle between 00 and 10. Note that D2 is always 0 because it is the sign bit here.

A.5 Mixed-Mode Neuron Test Structure with a Memristor Based Synapse

In this test structure, memristors are used to build the synapse instead of resistors. Hence, they need to be first formed and programmed to a desired synaptic weight before the testing with the neuron can be performed. The pin out for this test structure is shown in Fig. 16. The description of these pins is given by Table 7 and the test plan for this test structure is given by Table 8. This test structure is located at column 4 and rows 5,6. It may be noted that for this test structure, the output will be in terms of the neuron's spike (Fre). Fig. 18 shows the simulation result for the forming and programming of the memristors in the synapse. Both of the memristors are first 'formed' from their initial state. Later, both of them are programmed to their HRS state. Then, M_p is programmed back to LRS, whereas M_n stays at HRS, thus implementing a positive synaptic weight. Fig. 17 shows the simulation result for the neuron that is driven by this memristive synapse. In this case, the reference voltages applied are Vbias1=0V and Vbias2-3=1.2V, thus leading to an accumulation of '001' per each Fpre cycle. Note that here the result is similar to that of the resistive synapse case of Fig. 9. This demonstrates the operation of the memristive synapse with the neuron circuit. Further testing may be performed on this test structure by changing the reference voltages of the neuron and/or the memristive synaptic weight.

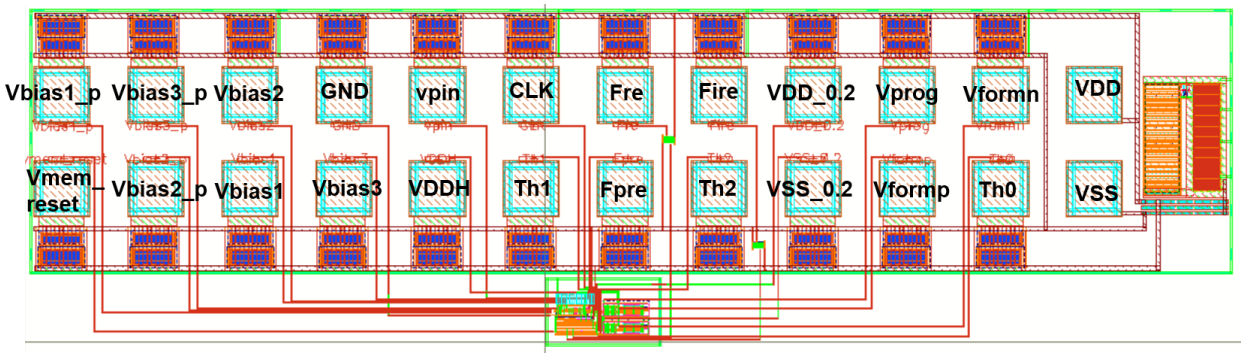

Figure 16: Pin out for the mixed-mode neuron test structure with memristor based synapse, connected to the probe pads.

Table 7: The pin description for the mixed-mode neuron test structure with memristor based synapse.

Pad Number	Pin Name	Pin type	Pin Description
1	VDD	Digital Input	Power Rail
2	Vformn	Digital Input	Forming Pin for M_n (3.3V)
3	Vprog	Digital Input	Programming Pin (3.3V)
4	VDD_0.2	Analog DC Input	Power rail for synapse input
5	Fire	Digital Output	Delayed Neuron Spike
6	Fre	Digital Output	Neuron Spike
7	CLK	Digital Pulse	Clock Signal
8	vpin	Digital Input	Supply Rail - 1.2V
9	GND	Analog DC Input	Mid-Rail Voltage
10	Vbias2	Analog DC Input	Reference Voltage
11	Vbias3_p	Analog DC Input	Reference Voltage
12	Vbias1_p	Analog DC Input	Reference Voltage
13	vmem_reset	Digital Pulse	Registers' RESET
14	Vbias2_p	Analog DC Input	Reference Voltage
15	Vbias1	Analog DC Input	Reference Voltage
16	Vbias3	Analog DC Input	Reference Voltage
17	VDDH	Digital Input	Supply Rail - 3.3V
18	Th1	Digital Input	Digital Threshold Bit
19	Fpre	Digital Input	Input Control
20	Th2	Digital Input	Digital Threshold Bit
21	VSS_0.2	Analog DC Input	Power rail for synapse input
22	Vformp	Digital Input	Forming Pin for M_p (3.3V)
23	Th0	Digital Input	Digital Threshold Bit
24	VSS	Digital Input	Power Rail

Table 8: The suggested test plan for the mixed-mode neuron test structure with memristor based synapse.

Pad Number	Pin Name	Pin Bias
1	VDD	1.2V
2	Vformn	3.3V when forming/programming to LRS
3	Vprog	3.3V when programming to HRS (along with Vform)
4	VDD_0.2	1.2V
5	Fire	Output
6	Fre	Output
7	CLK	Clock of any frequency with quick edges
8	vpin	1.2V
9	GND	0.6V
10	Vbias2	apply either VDD or VSS
11	Vbias3_p	0V
12	Vbias1_p	0V
13	vmem_reset	Apply a one time RESET pulse at startup
14	Vbias2_p	0V
15	Vbias1	apply either VDD or VSS
16	Vbias3	apply either VDD or VSS
17	VDDH	3.3V
18	Th1	1.2V
19	Fpre	Digital Pulse (half frequency of CLK)
20	Th2	0V
21	VSS_0.2	0V
22	Vformp	3.3V when forming/programming to LRS
23	Th0	1.2V
24	VSS	0V

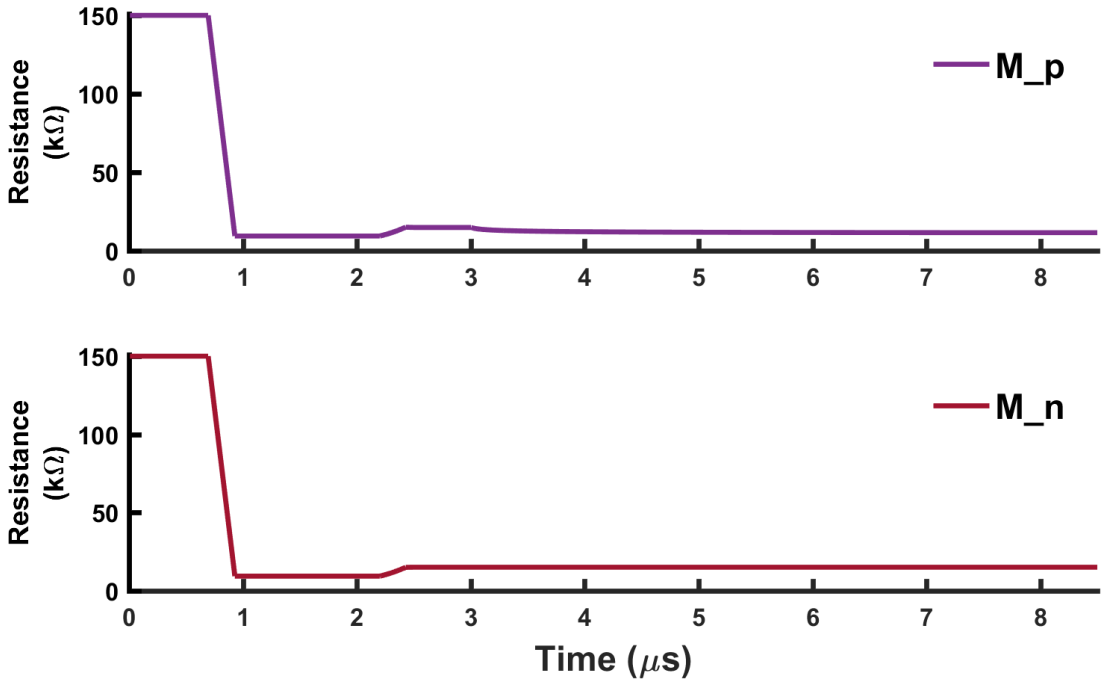

Figure 17: Simulation result showing the 'forming' step for the memristors from their initial state, followed by their programming steps.

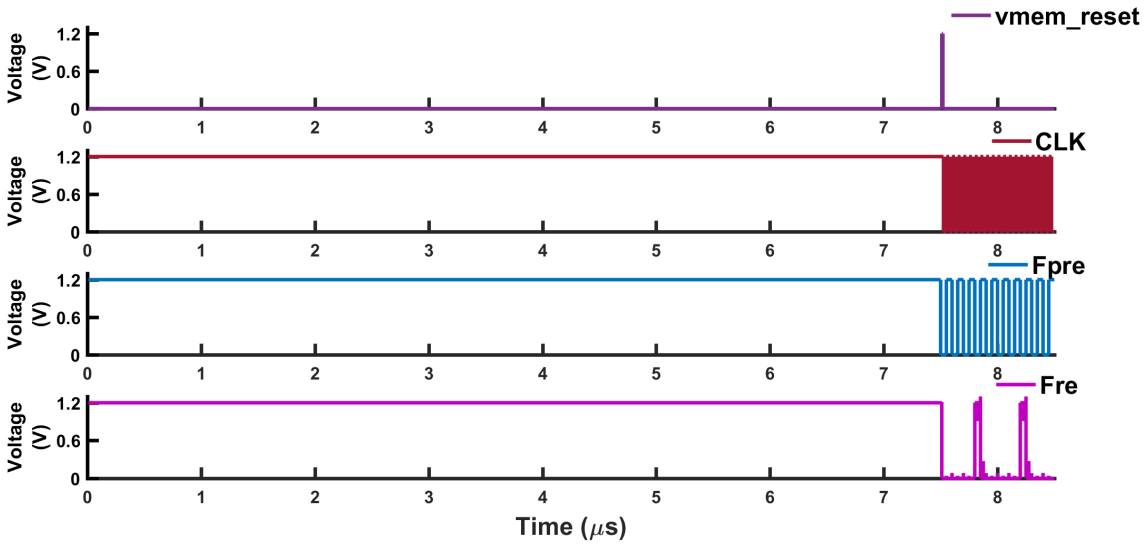

Figure 18: Simulation result showing the accumulation and spiking in the neuron connected to the memristive synapse.

B Python Code for Memristive Crossbar Based Pattern Recognition Application

In this section, the Python code for modeling the memristor based crossbar (discussed in Chapter 5) is presented. This code first instantiates a crossbar as an array of memristors (in this case a twin array because at each crosspoint there are two memristors). Later, the learning algorithm for the crosspoint synapses, followed by testing (for accuracy) is written. While the training runs, testing is interlaced with it and the accuracy is plotted as the epochs progress. Lastly, the learnt synaptic weights in the crossbar are also plotted. Comments in the code provide further explanation for the functioning of the corresponding part of the code. The code is as follows:

```
#Import necessary packages
import xlrd
import numpy as np
import matplotlib.pyplot as plt


#Memristor device parameters
HRS = 12000
LRS = 2500
Vtp = 0.6
Vtn = -0.6
tsw_p = 1e-6
tsw_n = 1e-6


theta_HRS = 0.85
beta_HRS = 0.07
theta_LRS = 1.6
beta_LRS=0.07
CLRS =  1
```

```
CHRS =  1
P_LRS =  2
P_HRS =  2

#Array of clock periods.
delt_arr = [200e-09, 40e-09, 20e-09, 13.33e-09, 10e-09, 5e-09]

#Variables used in memristor switching model
#Initial device resistance
Rinit = HRS
Rm = Rinit
Rm_tmp = 0

#Net positive voltages possible across synapse during STDP
vp1=1.0
vp2=0.9
vp3=0.8
vp4=0.7

#Net negative voltages possible across synapse during STDP
vn1=-1.0
vn2=-0.9
vn3=-0.8
vn4=-0.7

#Paramter needed for Memristor Switching Model
delR = HRS-LRS

#Memristor Switching Model
def memr_update(Rm, v, delt):
```

```
    if (v >= Vtp and Rm != LRS):
        Rm_tmp = Rm-delt*CLRS*(delR/tsw_p)*(((v-Vtp)/Vtp)**P_LRS)
            /(1+np.exp((theta_LRS*LRS-Rm)/(delR)/beta_LRS))
        if (Rm_tmp <= LRS):
            Rm_tmp = LRS

    elif (v <= Vtn and Rm != HRS):
        Rm_tmp = Rm + delt*CHRS*(delR/tsw_n)*(((v-Vtn)/Vtn)**P_HRS
            )/(1+np.exp((Rm-theta_HRS*HRS)/(delR)/beta_HRS))
        if (Rm_tmp >= HRS):
            Rm_tmp = HRS

    else:
        Rm_tmp = Rm
    Rm = Rm_tmp
    return Rm;

#'Tuning' parameter of neuron. Value chosen based on range of
    currents seen at neuron inputs. Can change this to adjust
    encoding rate/property
imax=6.2e-03
#Neuron model. This is inline with the concept of 'encoding' input
    current into digital values
def digi_neu(itot,vmem):
    if (itot>0 and itot<0.60*imax):
        vmem = 0
    if (itot>=0.60*imax and itot<0.65*imax):
        vmem = 1
    if (itot>=0.65*imax and itot<0.70*imax):
        vmem = 2
```

```
    if (itot >=0.70*imax and itot <0.75*imax):
        vmem=3
    if (itot >= 0.75*imax and itot <0.80*imax):
        vmem = 4
    if (itot >= 0.80*imax and itot <0.85*imax):
        vmem = 5
    if (itot >=0.85*imax and itot <0.90*imax):
        vmem = 6
    if (itot >=0.90*imax):
        vmem = 7
    if (itot <= 0):
        vmem = -1
    return vmem;

#Initializing crossbar array with initial values for Mp and Mn
mp_arr = [[Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,
    Rinit]]
for a in range(63):
    mp_arr.append([Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit
        ,Rinit,Rinit])
mn_arr = [[Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,
    Rinit]]
for a in range(63):
    mn_arr.append([Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit,Rinit
        ,Rinit,Rinit])

#Array containing Geff of all crossbar synapses
geff_arr=np.zeros((64,10))

#File containing training dataset
```

```
#Dataset elements modified from original as described in Section
   5.2
#Input downgraded values 7 thru 4 are identified here as 4 thru 1
   and 3 thru 0 as -1 thru -4 for convenience
book = xlrd.open_workbook('uci_new_try_tr1.xlsx')
sheet = book.sheet_by_name('Sheet1')
ip_arr = [[sheet.cell_value(r, c) for c in range(sheet.ncols)] for
    r in range(sheet.nrows)]

#File containing testing dataset
book2 = xlrd.open_workbook('uci_new_try_tst1.xlsx')
sheet2 = book2.sheet_by_name('Sheet1')
tes_arr = [[sheet2.cell_value(r, c) for c in range(sheet2.ncols)]
   for r in range(sheet2.nrows)]

epchs=[]
peracc=[]
count=0

#No. of epochs of training
epochs = 5

#Parameter to choose clock period. 20ns (50MHz) in this case
t=2

#Start of training
for e in range(epochs):
    for a in range(3823): #Iterating through each traning set
       pattern
        count += 1
```

```
if(ip_arr[a][64]==0): #Checking the label of the given
    pattern
     for b in range(64): #Performing STDP on corresponding
        column of synapses
         if (ip_arr[a][b]==4):
             mp_arr[b][0] = memr_update(mp_arr[b][0],vp1,
                delt_arr[t])
             mn_arr[b][0] = memr_update(mn_arr[b][0],vn1,
                delt_arr[t])
         elif (ip_arr[a][b]==3):
             mp_arr[b][0] = memr_update(mp_arr[b][0],vp2,
                delt_arr[t])
             mn_arr[b][0] = memr_update(mn_arr[b][0],vn2,
                delt_arr[t])
         elif (ip_arr[a][b]==2):
             mp_arr[b][0] = memr_update(mp_arr[b][0],vp3,
                delt_arr[t])
             mn_arr[b][0] = memr_update(mn_arr[b][0],vn3,
                delt_arr[t])
         elif (ip_arr[a][b]==1):
             mp_arr[b][0] = memr_update(mp_arr[b][0],vp4,
                delt_arr[t])
             mn_arr[b][0] = memr_update(mn_arr[b][0],vn4,
                delt_arr[t])
         elif (ip_arr[a][b]==-1):
             mp_arr[b][0] = memr_update(mp_arr[b][0],vn4,
                delt_arr[t])
             mn_arr[b][0] = memr_update(mn_arr[b][0],vp4,
                delt_arr[t])
         elif (ip_arr[a][b]==-2):
```

```
                mp_arr[b][0] = memr_update(mp_arr[b][0],vn3,
                    delt_arr[t])
                mn_arr[b][0] = memr_update(mn_arr[b][0],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][0] = memr_update(mp_arr[b][0],vn2,
                    delt_arr[t])
                mn_arr[b][0] = memr_update(mn_arr[b][0],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][0] = memr_update(mp_arr[b][0],vn1,
                    delt_arr[t])
                mn_arr[b][0] = memr_update(mn_arr[b][0],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==1):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vp1,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vp2,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vn2,
                    delt_arr[t])
            elif (ip_arr[a][b]==2):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vp3,
                    delt_arr[t])
```

```
                mn_arr[b][1] = memr_update(mn_arr[b][1],vn3,
                    delt_arr[t])
            elif (ip_arr[a][b]==1):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vp4,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vn4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-1):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vn4,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vp4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-2):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vn3,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][1] = memr_update(mp_arr[b][1],vn2,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][1] = memr_update(mp_arr[b][1],vn1,
                    delt_arr[t])
                mn_arr[b][1] = memr_update(mn_arr[b][1],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==2):
        for b in range(64):
```

```
if (ip_arr[a][b]==4):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vp1,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vn1,
        delt_arr[t])
elif (ip_arr[a][b]==3):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vp2,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vn2,
        delt_arr[t])
elif (ip_arr[a][b]==2):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vp3,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vn3,
        delt_arr[t])
elif (ip_arr[a][b]==1):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vp4,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vn4,
        delt_arr[t])
elif (ip_arr[a][b]==-1):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vn4,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vp4,
        delt_arr[t])
elif (ip_arr[a][b]==-2):
    mp_arr[b][2] = memr_update(mp_arr[b][2],vn3,
        delt_arr[t])
    mn_arr[b][2] = memr_update(mn_arr[b][2],vp3,
        delt_arr[t])
```

```
            elif (ip_arr[a][b]==-3):
                mp_arr[b][2] = memr_update(mp_arr[b][2],vn2,
                    delt_arr[t])
                mn_arr[b][2] = memr_update(mn_arr[b][2],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][2] = memr_update(mp_arr[b][2],vn1,
                    delt_arr[t])
                mn_arr[b][2] = memr_update(mn_arr[b][2],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==3):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vp1,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vp2,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vn2,
                    delt_arr[t])
            elif (ip_arr[a][b]==2):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vp3,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vn3,
                    delt_arr[t])
            elif (ip_arr[a][b]==1):
```

```
                mp_arr[b][3] = memr_update(mp_arr[b][3],vp4,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vn4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-1):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vn4,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vp4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-2):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vn3,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][3] = memr_update(mp_arr[b][3],vn2,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][3] = memr_update(mp_arr[b][3],vn1,
                    delt_arr[t])
                mn_arr[b][3] = memr_update(mn_arr[b][3],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==4):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][4] = memr_update(mp_arr[b][4],vp1,
                    delt_arr[t])
```

```
            mn_arr[b][4] = memr_update(mn_arr[b][4],vn1,
                delt_arr[t])
        elif (ip_arr[a][b]==3):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vp2,
                delt_arr[t])
            mn_arr[b][4] = memr_update(mn_arr[b][4],vn2,
                delt_arr[t])
        elif (ip_arr[a][b]==2):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vp3,
                delt_arr[t])
            mn_arr[b][4] = memr_update(mn_arr[b][4],vn3,
                delt_arr[t])
        elif (ip_arr[a][b]==1):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vp4,
                delt_arr[t])
            mn_arr[b][4] = memr_update(mn_arr[b][4],vn4,
                delt_arr[t])
        elif (ip_arr[a][b]==-1):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vn4,
                delt_arr[t])
            mn_arr[b][4] = memr_update(mn_arr[b][4],vp4,
                delt_arr[t])
        elif (ip_arr[a][b]==-2):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vn3,
                delt_arr[t])
            mn_arr[b][4] = memr_update(mn_arr[b][4],vp3,
                delt_arr[t])
        elif (ip_arr[a][b]==-3):
            mp_arr[b][4] = memr_update(mp_arr[b][4],vn2,
                delt_arr[t])
```

```
                mn_arr[b][4] = memr_update(mn_arr[b][4],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][4] = memr_update(mp_arr[b][4],vn1,
                    delt_arr[t])
                mn_arr[b][4] = memr_update(mn_arr[b][4],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==5):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vp1,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vp2,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vn2,
                    delt_arr[t])
            elif (ip_arr[a][b]==2):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vp3,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vn3,
                    delt_arr[t])
            elif (ip_arr[a][b]==1):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vp4,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vn4,
                    delt_arr[t])
```

```
            elif (ip_arr[a][b]==-1):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vn4,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vp4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-2):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vn3,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][5] = memr_update(mp_arr[b][5],vn2,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][5] = memr_update(mp_arr[b][5],vn1,
                    delt_arr[t])
                mn_arr[b][5] = memr_update(mn_arr[b][5],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==6):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][6] = memr_update(mp_arr[b][6],vp1,
                    delt_arr[t])
                mn_arr[b][6] = memr_update(mn_arr[b][6],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
```

```
            mp_arr[b][6] = memr_update(mp_arr[b][6],vp2,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vn2,
                delt_arr[t])
        elif (ip_arr[a][b]==2):
            mp_arr[b][6] = memr_update(mp_arr[b][6],vp3,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vn3,
                delt_arr[t])
        elif (ip_arr[a][b]==1):
            mp_arr[b][6] = memr_update(mp_arr[b][6],vp4,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vn4,
                delt_arr[t])
        elif (ip_arr[a][b]==-1):
            mp_arr[b][6] = memr_update(mp_arr[b][6],vn4,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vp4,
                delt_arr[t])
        elif (ip_arr[a][b]==-2):
            mp_arr[b][6] = memr_update(mp_arr[b][6],vn3,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vp3,
                delt_arr[t])
        elif (ip_arr[a][b]==-3):
            mp_arr[b][6] = memr_update(mp_arr[b][6],vn2,
                delt_arr[t])
            mn_arr[b][6] = memr_update(mn_arr[b][6],vp2,
                delt_arr[t])
        else:
```

```
                mp_arr[b][6] = memr_update(mp_arr[b][6],vn1,
                    delt_arr[t])
                mn_arr[b][6] = memr_update(mn_arr[b][6],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==7):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vp1,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vp2,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vn2,
                    delt_arr[t])
            elif (ip_arr[a][b]==2):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vp3,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vn3,
                    delt_arr[t])
            elif (ip_arr[a][b]==1):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vp4,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vn4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-1):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vn4,
                    delt_arr[t])
```

```
                mn_arr[b][7] = memr_update(mn_arr[b][7],vp4,
                    delt_arr[t])
            elif (ip_arr[a][b]==-2):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vn3,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][7] = memr_update(mp_arr[b][7],vn2,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][7] = memr_update(mp_arr[b][7],vn1,
                    delt_arr[t])
                mn_arr[b][7] = memr_update(mn_arr[b][7],vp1,
                    delt_arr[t])

    if(ip_arr[a][64]==8):
        for b in range(64):
            if (ip_arr[a][b]==4):
                mp_arr[b][8] = memr_update(mp_arr[b][8],vp1,
                    delt_arr[t])
                mn_arr[b][8] = memr_update(mn_arr[b][8],vn1,
                    delt_arr[t])
            elif (ip_arr[a][b]==3):
                mp_arr[b][8] = memr_update(mp_arr[b][8],vp2,
                    delt_arr[t])
                mn_arr[b][8] = memr_update(mn_arr[b][8],vn2,
                    delt_arr[t])
```

```
elif (ip_arr[a][b]==2):
    mp_arr[b][8] = memr_update(mp_arr[b][8],vp3,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vn3,
        delt_arr[t])
elif (ip_arr[a][b]==1):
    mp_arr[b][8] = memr_update(mp_arr[b][8],vp4,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vn4,
        delt_arr[t])
elif (ip_arr[a][b]==-1):
    mp_arr[b][8] = memr_update(mp_arr[b][8],vn4,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vp4,
        delt_arr[t])
elif (ip_arr[a][b]==-2):
    mp_arr[b][8] = memr_update(mp_arr[b][8],vn3,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vp3,
        delt_arr[t])
elif (ip_arr[a][b]==-3):
    mp_arr[b][8] = memr_update(mp_arr[b][8],vn2,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vp2,
        delt_arr[t])
else:
    mp_arr[b][8] = memr_update(mp_arr[b][8],vn1,
        delt_arr[t])
    mn_arr[b][8] = memr_update(mn_arr[b][8],vp1,
        delt_arr[t])
```

```
if(ip_arr[a][64]==9):
    for b in range(64):
        if (ip_arr[a][b]==4):
            mp_arr[b][9] = memr_update(mp_arr[b][9],vp1,
               delt_arr[t])
            mn_arr[b][9] = memr_update(mn_arr[b][9],vn1,
               delt_arr[t])
        elif (ip_arr[a][b]==3):
            mp_arr[b][9] = memr_update(mp_arr[b][9],vp2,
               delt_arr[t])
            mn_arr[b][9] = memr_update(mn_arr[b][9],vn2,
               delt_arr[t])
        elif (ip_arr[a][b]==2):
            mp_arr[b][9] = memr_update(mp_arr[b][9],vp3,
               delt_arr[t])
            mn_arr[b][9] = memr_update(mn_arr[b][9],vn3,
               delt_arr[t])
        elif (ip_arr[a][b]==1):
            mp_arr[b][9] = memr_update(mp_arr[b][9],vp4,
               delt_arr[t])
            mn_arr[b][9] = memr_update(mn_arr[b][9],vn4,
               delt_arr[t])
        elif (ip_arr[a][b]==-1):
            mp_arr[b][9] = memr_update(mp_arr[b][9],vn4,
               delt_arr[t])
            mn_arr[b][9] = memr_update(mn_arr[b][9],vp4,
               delt_arr[t])
        elif (ip_arr[a][b]==-2):
```

```
                mp_arr[b][9] = memr_update(mp_arr[b][9],vn3,
                    delt_arr[t])
                mn_arr[b][9] = memr_update(mn_arr[b][9],vp3,
                    delt_arr[t])
            elif (ip_arr[a][b]==-3):
                mp_arr[b][9] = memr_update(mp_arr[b][9],vn2,
                    delt_arr[t])
                mn_arr[b][9] = memr_update(mn_arr[b][9],vp2,
                    delt_arr[t])
            else:
                mp_arr[b][9] = memr_update(mp_arr[b][9],vn1,
                    delt_arr[t])
                mn_arr[b][9] = memr_update(mn_arr[b][9],vp1,
                    delt_arr[t])

#Evaluating Geff array
for p in range(64):
    for q in range(10):
        geff_arr[p][q]=(1/mp_arr[p][q]-1/mn_arr[p][q])

#Start of testing
if (count<= 2000): #Chosen such that we test more
   frequently initially
    if(count%150==0): #Frequency with which testing is
       performed when epochs are running
        epchs.append(count)
        tot = 0
        acc = 0
        for m in range(1797): #Iterating through each
           testing set pattern
```

```
tot +=1
cnt = 0
itot = np.zeros(10)
vmem_arr = np.zeros(10)
#Testing by applying input neuron spikes to
    columns
#Spikes reversed for negative input array
    elements
#Neuron with array element value +1/-1 spikes
    first followed by others with 2 cycles of
    delay between each
#Output neuron samples column currents during
    the cycle when +4/-4 neurons apply +0.4V
    /-0.4V on Mp memristor and vice versa on Mn
#By this time +3/-3 neurons apply +0.2V/-0.2V
    on Mp and vice versa for Mn (because of 2
    cycle delay between them)
#Spikes of +2/-2 and +1/-1 neurons would have
    finished and therefore they do not
    contribute for synapse currents into
    columns
#This scheme is chosen to reduce column
    currents during testing without
    compromising test accuracy
#This scheme may be changed by changing delay
    between input neurons' spikes and/or when
    the output neuron samples column currents
for n in range(10):
    for q in range(64):
        if(tes_arr[m][q]==4):
```

```
                itot[n]+=0.4*geff_arr[q][n]
            elif(tes_arr[m][q]==3):
                itot[n]+=0.2*geff_arr[q][n]
            elif(tes_arr[m][q]==2):
                itot[n]+=0*geff_arr[q][n]
            elif(tes_arr[m][q]==1):
                itot[n]+=0*geff_arr[q][n]
            elif(tes_arr[m][q]==-4):
                itot[n]+=-0.4*geff_arr[q][n]
            elif(tes_arr[m][q]==-3):
                itot[n]+=-0.2*geff_arr[q][n]
            elif(tes_arr[m][q]==-2):
                itot[n]+=-0*geff_arr[q][n]
            else:
                itot[n]+=-0*geff_arr[q][n]

#Digitization of column currents in output
    neurons
for d in range(10):
    vmem_arr[d] = digi_neu(itot[d],vmem_arr[d
        ])
#WTA logic at output to determine highest
    digital value/'winning' neuron
for r in range(10):
    if (vmem_arr[r] == max(vmem_arr)):
        cnt+=1
#Determining if 'winning' neuron label matches
    with input label
if (cnt==1):
    if (tes_arr[m][64]==np.argmax(vmem_arr)):
```

```
                        acc += 1
                    else:
                        acc += 0
                else:
                    acc += 0

            per_acc = (acc*100)/tot #Accuracy
            peracc.append(per_acc) #Accuracy array
    else:
        if(count%500==0): #Reduced frequency of testing after
            some 'sufficient' training
            epchs.append(count)
            tot = 0
            acc = 0
            for m in range(1797):
                tot +=1
                cnt = 0
                itot = np.zeros(10)
                vmem_arr = np.zeros(10)
                for n in range(10):
                    for q in range(64):
                        if(tes_arr[m][q]==4):
                            itot[n]+=0.4*geff_arr[q][n]
                        elif(tes_arr[m][q]==3):
                            itot[n]+=0.2*geff_arr[q][n]
                        elif(tes_arr[m][q]==2):
                            itot[n]+=0*geff_arr[q][n]
                        elif(tes_arr[m][q]==1):
                            itot[n]+=0*geff_arr[q][n]
                        elif(tes_arr[m][q]==-4):
```

```
                    itot[n]+=-0.4*geff_arr[q][n]
                elif(tes_arr[m][q]==-3):
                    itot[n]+=-0.2*geff_arr[q][n]
                elif(tes_arr[m][q]==-2):
                    itot[n]+=-0*geff_arr[q][n]
                else:
                    itot[n]+=-0*geff_arr[q][n]

            for d in range(10):
                vmem_arr[d] = digi_neu(itot[d],vmem_arr[d
                    ])
            for r in range(10):
                if (vmem_arr[r] == max(vmem_arr)):
                    cnt+=1
            if (cnt==1):
                if (tes_arr[m][64]==np.argmax(vmem_arr)):
                    acc += 1
                else:
                    acc += 0
            else:
                acc += 0

        per_acc = (acc*100)/tot
        peracc.append(per_acc)

#Plotting accuracy curve as training progresses
plt.plot(epchs,peracc,'o-',color='blue',linewidth=2.2)
plt.ylim(0,100)
plt.xlim(0,18000)
plt.grid(True)
```

```
plt.xlabel('No. of Learning Epochs',fontweight='bold',fontsize=12)
plt.ylabel('Recognition Rate (%)',fontweight='bold',fontsize=12)
plt.xticks(np.arange(0,19116,3823),('0','1','2','3','4','5'))

#Print column currents
print(itot)
#Print digitized values in output neurons
print(vmem_arr)
#Print accuracy of final test
print(per_acc)

#Creating arrays to store learnt weights as 8x8 arrays for
   plotting them and visually seeing the learnt weight patterns
im0=np.zeros((8,8))
im1=np.zeros((8,8))
im2=np.zeros((8,8))
im3=np.zeros((8,8))
im4=np.zeros((8,8))
im5=np.zeros((8,8))
im6=np.zeros((8,8))
im7=np.zeros((8,8))
im8=np.zeros((8,8))
im9=np.zeros((8,8))

#Converting learnt Geff array into 8x8 patterns
for a in range(8):
    for b in range(8):
        im0[a][b]=1e06*geff_arr[b+(a*8)][0]

for a in range(8):
```

```
    for b in range(8):
        im1[a][b]=1e06*geff_arr[b+(a*8)][1]

for a in range(8):
    for b in range(8):
        im2[a][b]=1e06*geff_arr[b+(a*8)][2]

for a in range(8):
    for b in range(8):
        im3[a][b]=1e06*geff_arr[b+(a*8)][3]

for a in range(8):
    for b in range(8):
        im4[a][b]=1e06*geff_arr[b+(a*8)][4]

for a in range(8):
    for b in range(8):
        im5[a][b]=1e06*geff_arr[b+(a*8)][5]

for a in range(8):
    for b in range(8):
        im6[a][b]=1e06*geff_arr[b+(a*8)][6]

for a in range(8):
    for b in range(8):
        im7[a][b]=1e06*geff_arr[b+(a*8)][7]

for a in range(8):
    for b in range(8):
        im8[a][b]=1e06*geff_arr[b+(a*8)][8]
```

```
for a in range(8):
    for b in range(8):
        im9[a][b]=1e06*geff_arr[b+(a*8)][9]

#Plotting the learnt weights as image patterns for inspection
fig,ax=plt.subplots()
img=ax.imshow(im0)
fig.colorbar(img, ax=ax)

fig,ax=plt.subplots()
img=ax.imshow(im1)
fig.colorbar(img, ax=ax)

fig,ax=plt.subplots()
img=ax.imshow(im2)
fig.colorbar(img, ax=ax)

fig,ax=plt.subplots()
img=ax.imshow(im3)
fig.colorbar(img, ax=ax)

fig,ax=plt.subplots()
img=ax.imshow(im4)
fig.colorbar(img, ax=ax)

fig,ax=plt.subplots()
img=ax.imshow(im5)
fig.colorbar(img, ax=ax)
```

```
fig ,ax=plt.subplots()
img=ax.imshow(im6)
fig.colorbar(img, ax=ax)

fig ,ax=plt.subplots()
img=ax.imshow(im7)
fig.colorbar(img, ax=ax)

fig ,ax=plt.subplots()
img=ax.imshow(im8)
fig.colorbar(img, ax=ax)

fig ,ax=plt.subplots()
img=ax.imshow(im9)
fig.colorbar(img, ax=ax)
```

Vita

Sagarvarma Sayyaparaju received the Bachelor of Technology (B.Tech) degree in Electronics and Communication Engineering from the Indian Institute of Technology Roorkee, India, in 2016. He has been working towards his PhD degree in Electrical Engineering at the University of Tennessee, Knoxville since Fall 2016. His research interests include neuromorphic computing, nanoelectronic circuit design, emerging devices and CMOS VLSI circuit design.