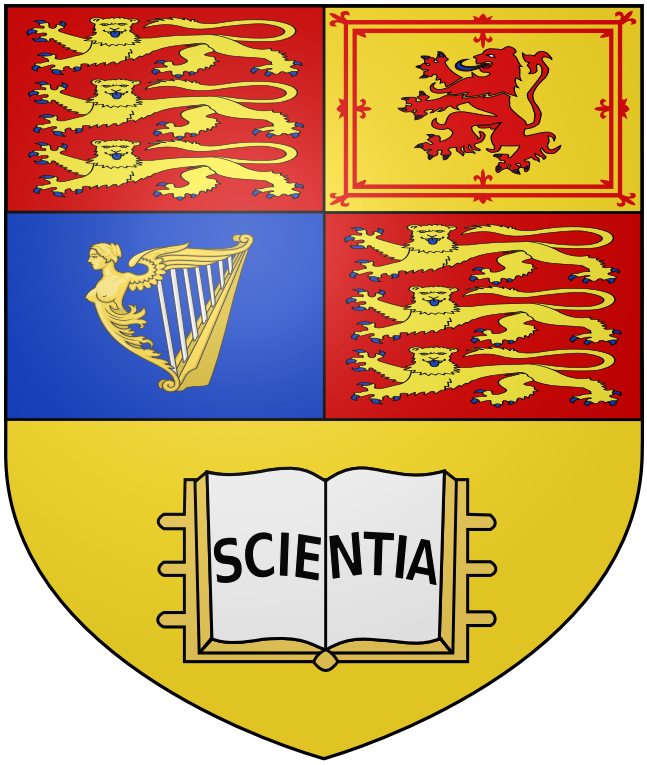
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2021



Project Title:	**A memristive machine learning processor**
Student:	**Tej Minar Sanghavi**
CID:	**01348755**
Course:	**EEE4**
Project Supervisor:	**Dr Christos Papavassiliou**
Second Marker:	**Dr Timothy Constandinou**

I affirm that I have submitted, or will submit, electronic copies of my final year project report to both Blackboard and the EEE coursework submission system.
I affirm that the two copies of the report are identical.
I affirm that I have provided explicit references for all material in my Final Report which is not authored by me and represented as my own work.

Tej Minar Sanghavi (2021)

Acknowledgement

I would like to thank my supervisor Dr Christos Papavassiliou and his PhD student Adil Malik for their invaluable guidance, support and encouragement throughout my project. I would like to thank the faculty at Imperial College and my teachers from school who have mentored me and pushed me to achieving feats I could have only ever dreamt of. Lastly, I would like to thank my friends and family for their love and support. My sincerest gratitude goes out to my family for all they have done to support me throughout this voyage. It has been clear at times, rough and choppy at others, ever changing, but they have been a constant in my life and never fail to bring me to shore.

Abstract

The announcement of the world's first memristor in 2008 heralded a new era in the field of neuromorphic engineering. These non-volatile devices have a characteristic that allows their resistance to be programmed and demonstrate synaptic-like properties. Since then, researchers have made great efforts in developing memristive based system for machine learning applications. However, what has deterred researchers from realising a fully neural architecture is the unpredictable behaviour of sneak-currents in selectorless memritor crossbar arrays. Using these arrays, this project computationally replicates a software neural network model to prove that they can indeed perform machine learning computations reliably. The results drawn from investigating its performance furthermore, challenges the widely recognized opinion that sneak currents are detrimental to crossbars and lays the foundation for designing such architectures that have revolutionary potential.

Contents

Chapter 1

Introduction

Traditional Von Neumann computing architectures physically separate processing and memory blocks [1]. The latency in the data flow between the two modules [2] creates a bottleneck that limits the performance of such a system while inefficient data paths expend energy. In [3] it was shown that the rate of improvement in the overall performance of a computing system will eventually be limited by memory access speeds since the rate of improvement in processor speeds exceeds the rate of improvement in memory access speeds. During the past decade, the rapid growth in Artificial Intelligence and Machine Learning in particular has highlighted this bottleneck in performance during the training and inference of deep neural networks (DNNs) where the fetching and updating of tens of millions of parameters (synaptic weights) leads to weeks of computation time and many kilowatts of power consumption limiting the future ability to scale such networks to sizes found in human brains ($> 10^{15}$ synapses).

The industry has made efforts to use a large number of parallel processors, namely graphics processing units (GPU) [4] or processors customized to efficiently process DNNs [5], but it is impossible to completely address the issue of inefficient data flows that consume time and energy. An obvious idea, would be to physically reduce the distance between the memory and the processing unit as much as possible and this gave rise to the near-memory computing architecture [6]. The architecture relies on through silicon via technology (TSV) that allows dies to be stacked vertically bringing memory and processing even closer. Nevertheless, this has not fundamentally eliminated the physical isolation between the two blocks and is limited by the TSV interconnect density that limits data bandwidth.

The unique capability of synapses to store and process information combined with the human brain's highly parallel structure has for many decades inspired research into developing hardware systems that try to emulate this giving rise to the field of Neuromorphic Engineering [7]. Recently, neuo-inspired integrated circuits (ICs) or hardware accelerators based on CMOS technology have been designed and have achieved a significant improvement in computing performance [8]. However, CMOS technology usually requires redundant transistors to build artificial synapses and neurons and are far from achieving the efficiency of the brain which operates at a low frequency of 10Hz and has a low energy consumption of just 20 W [9].

In 2008, researchers at HP Labs announced the fabrication of the first memristor [10]. These are devices whose resistive state can be modulated based on their flux-charge relationship giving them unique characteristics that allow them to behave like synapses - their resistive state can reflect data storage and computations can be performed on them. Since then, neuromorphic

research interests in these devices have boomed and various memristive devices have been reported in literature [11]. These devices typically fabricate memristors in a crossbar architecture [12] which offers high density, random access, large connectivity and most importantly performs multiply accumulate operations (which are the core of machine learning algorithms) without any additional hardware. However, these crossbar architectures are based on a 1-Transistor-1-Memristor structure which limits device density and thus limits a fully neural architecture from being realised.

Selectorless memristor crossbars (only memristors) face a non-ideality known as sneak-currents whose unpredictable nature has deterred researchers from investigating their use. This project aims at investigating whether this phenomenon prevents selectorless memristor crossbars arrays from reliably computing multiply accumulate operations by replicating a software model of a neural network to draw conclusions.

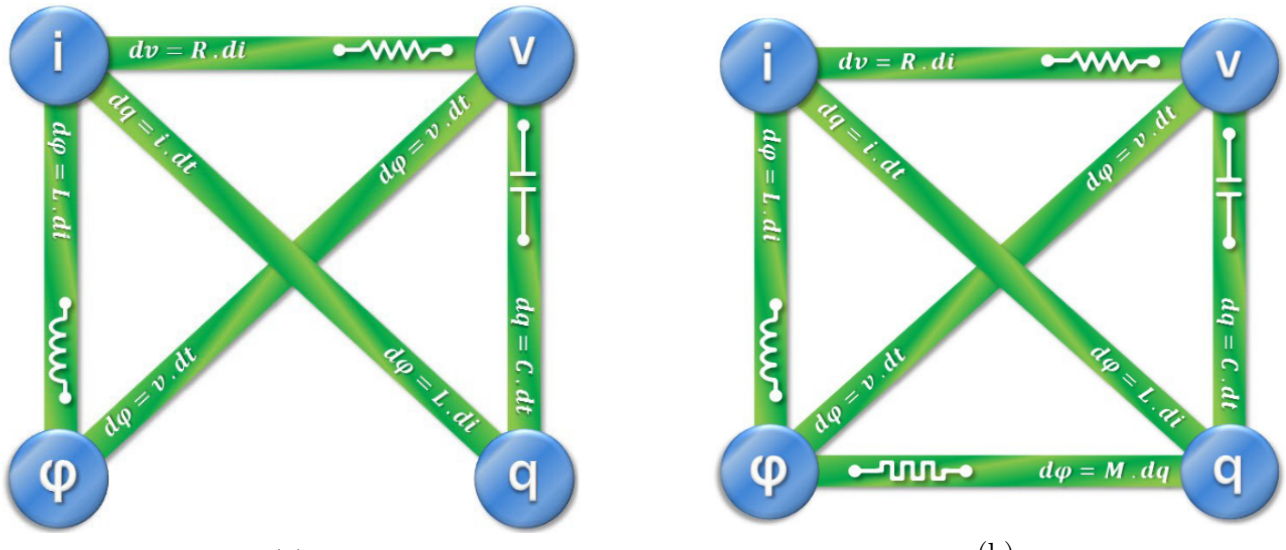
The order of the project is as follows: Chapter 2 provides the reader with the foundation to understand the behaviour of memristors and how they can be interfaced to a crossbar architecture, Chapter 3 practically demonstrates some of the theoretical concepts discussed, Chapter 4 introduces the software model that will be replicated to investigate whether reliable computations are reliable with a selectorless memristor crossbar and Chapter 5 describes its practical implementation. Finally, Chapter 6 investigates the performance of the architecture and discusses the various results and observations drawn.

Chapter 2

Background

2.1 Memristor - The missing link

In 1971 Leon Chua published a paper [13] that postulated the existence of a fourth fundamental circuit element that he termed 'Memristor' whose theorised behaviour is similar to a resistor with memory. Chua realised that out of the four fundamental circuit variables: current i, electrical charge q, voltage v and the magnetic flux ϕ - there existed relationships between any one variable and another in all cases except between ϕ and q. The relationship between voltage and current, voltage and charge, and current and flux are defined by passive two-terminal fundamental circuit elements resistor, capacitor and an inductor, respectively. While, the two other combinations are defined by Faraday's law. Faraday's law shows that current is the time derivative of electric charge and the voltage is the time derivative of the magnetic flux. Figure 2.1(a) shows the relationship between these parameters. The missing relation between



(a) (b)

Figure 2.1: The relation between the four fundamental circuit variables before (a) and after (b) Chua's paper. Note, the symbol of a memristor.

magnetic flux and electric charge is shown in Figure 2.1(b) and Chua argued that on grounds of symmetry and completeness the laws of magnetism and electricity allow for the existence of a fourth fundamental passive circuit element. Chua then suggested that the memristor has the

following qualitative properties:

- simple structure of just two terminals;
- the internal state variable is resistance;
- the device stores information on how much charge has passed through it, and in which direction, this information is encoded in its internal state variable; the resistance of the memristor increases if current passes in one direction, and decreases otherwise;
- the internal state variable is non-volatile;
- it is a passive device not requiring an external power supply.

2.2 Memristance

The relationship between flux and charge of a memristor can be expressed as a function of flux or charge depending on how it is controlled.

For a charge-controlled memristor,

$$\varphi \; = \; f(q) \tag{2.1}$$

differentiating yields,

$$\frac{d\varphi}{dt} \; = \; \frac{df(q)}{dq} \cdot \frac{dq}{dt} \tag{2.2}$$

since voltage is, $v(t) = \frac{d\varphi}{dt}$ and current is, $i(t) = \frac{dq}{dt}$ the equation can be rewritten as:

$$v(t) = M(q)i(t) \qquad where\ M(q) = \frac{df(q)}{dq} \tag{2.3}$$

$M(q)$ is the memristance with units Ohms similar to a resistor.

Similarly, for a flux controlled memristor,

$$q \; = \; f(\varphi) \tag{2.4}$$

differentiating yields,

$$\frac{dq}{dt} \; = \; \frac{df(\varphi)}{d\phi} \cdot \frac{d\varphi}{dt} \tag{2.5}$$

since voltage is, $v(t) = \frac{d\varphi}{dt}$ and current is, $i(t) = \frac{dq}{dt}$ the equation can be rewritten as:

$$v(t) = W(\varphi)i(t) \qquad where\ W(\varphi) = \frac{df(\varphi)}{d\varphi} \tag{2.6}$$

$W(\varphi)$ is the memductance with units Siemens similar to conductance

2.3 Characteristics of memristors

2.3.1 Flux-charge relation

The q-φ curve of a memristor has a a monotonically increasing characteristic with a shape dependant on the type of memristor and gradient given by the memristance M(q). One such curve is shown in Figure 2.2.

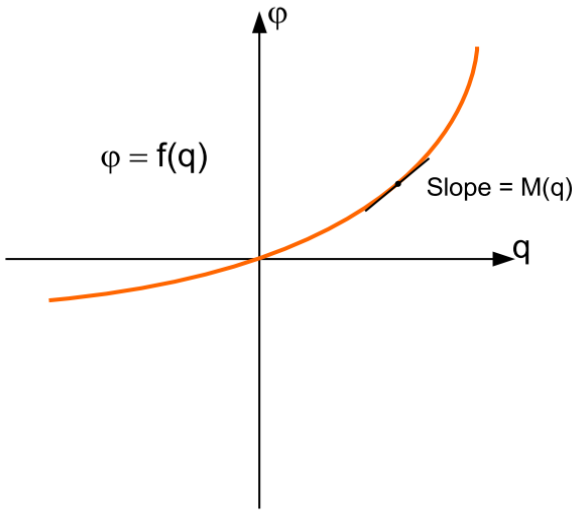


Figure 2.2: Charge-flux characteristics of a memristor

2.3.2 Current-voltage relation

An important characteristic of a memristor is that it exhibits a pinched hysteresis loop that crosses the origin when a periodic signal is applied to it as shown in Figure 2.3. Moreover, with higher frequencies the area of the hysteresis loop shrinks and as frequency approaches infinity the memristor exhibits a I-V relationship similar to a resistor.

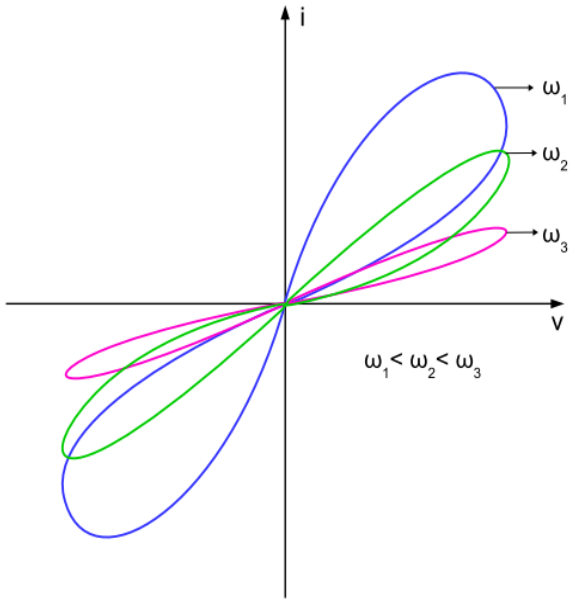


Figure 2.3: Current-voltage characteristic of a memristor. The pinched hysteresis loop shrinks with increasing the frequency

2.4 Memristive systems

Five years after his first paper on memristors, Chua published a paper [14] that generalised memristors to an interesting class of nonlinear dynamical systems called memristive systems.

The systems are characterised by a time-dependent state equation which links input and output of a system in the following way:

$$\dot{x} = f(x,u,t) \tag{2.7}$$

$$y = g(x,u,t)u \tag{2.8}$$

where x represents the internal state of the system, u and y represent the input and output to the system, and f and g are both continuous functions. The constituent relationships can be qualitatively described as: f governs how the system state changes based on current state, input and time while g defines the output of the system given its instantaneous state, input and time.

For a time-invariant memristive electronic device, the above equations can be written in either of two ways, depending on what the input to the system is considered to be: (a): for a charge controlled device, the input to the system is represented by current as in (2.9) and output is voltage; and (b): for a flux-controlled device, the input to the system is represented by voltage and output is current as in (2.10) :

$$\dot{x} = f(x,i), \qquad v = R(x,i)i \tag{2.9}$$

$$\dot{x} = f(x,v), \qquad i = G(x,v)v \tag{2.10}$$

For a dynamic memristive device, the internal state x is a critical n-dimension vector which encompasses all state-parameters of the system and can have a large number of distinct elements. It immediately becomes obvious that in this framework, the passive resistor is a subset of memristive systems and can be described as a time-invariant, 0-th order memristive device, where for example R = x, x is the static resistance of the device, R(x, i) = R and f = 0.

A dynamic memristive devices' signature behaviour is expressed as a pinched hysteresis loop (as seen in Figure 2.3), a behaviour which Chua claims defines memristive effects, irrespective of the mechanism that causes it.

2.5 Memristive Devices

A range of memristive devices have been reported in literature dating back back to more than 50 years ago. An important class of two-terminal memristive devices are based on ionic motion, which are built from a conductor/insulator/conductor thin-film stack and recent progress has led to fast, low-energy, high-endurance devices that can be scaled down to less than 10 nm, stacked in three dimensions and can be integrated with CMOS technology to form hybrid CMOS/memristor circuits with computing capabilities. A thorough overview of these devices can be found here [11] and a high level overview of important points will be mentioned below.

Various memristive devices require an inital *electroforming step* before their resistive switching properties can be observed and this consists of applying a voltage ramp to the device. A common first cycle switching I-V curve is illustrated in Figure 2.4a, where the electroforming voltage is higher than the subsequent SET and RESET voltage thresholds (device transition from high resistance state towards low resistance state is called a SET process, while the opposite is named RESET). Moreover, the complex interplay between the various forces within the memristor gives rise to a rich collection of resistive switching event types which can be grossly divided into two main branches: Bipolar switching - when opposite polarity electric stimuli are required to

achieve SET or RESET, respectively; and Unipolar switching - also referred to as non-polar switching, when the same polarity signal can be used to achieve either SET or RESET. These two switching mechanisms have several other peculiarities depending on what is the main driving force for charge within the device. These determine what the electrical behavior of the respective devices is and are classified in Figure 2.4b-e.

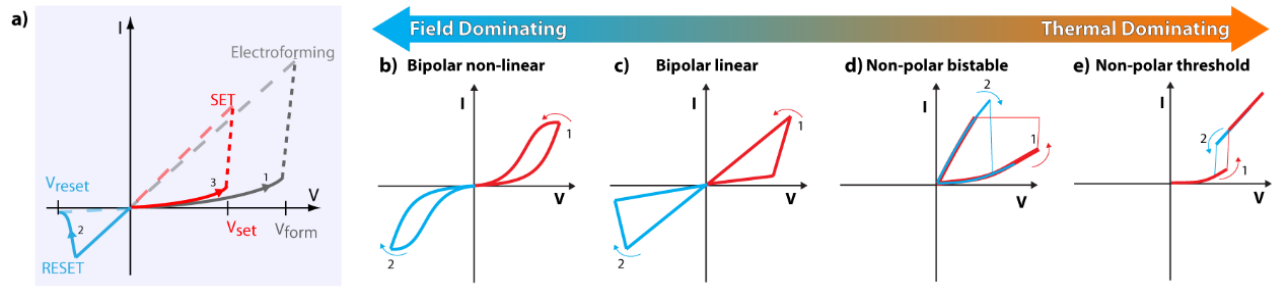


Figure 2.4: Different resistance switching types identified [15] a) Illustration of a common memristive device I-V curve with focus on electroforming, followed by first RESET and first SET, in order. Illustrations of resistive switching types b) Bipolar non-linear; c) Bipolar linear; d) Non-polar bistable; e) Non-polar threshold;

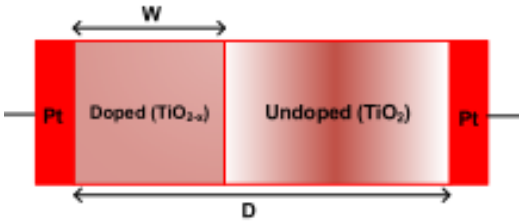
2.6 Memristor Models

To be able to design, analyze and simulate memristor based circuits and applications, a model is needed. In 2008, researchers from HP Lab's published a paper [10] that first established the link between nanoscale electronic devices and memristor theory.

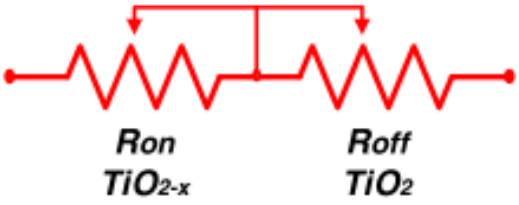
The remainder of this section will present an overview of HP Labs memristor and the model used to link it to memristor theory followed by summaries of other models and window functions that have been proposed in literature.

2.6.1 Linear Ion Drift model

HP Lab's memristor consists of a thin film of titanium dioxide (TiO2) semiconductor sandwiched between two platinum electrodes as shown in Figure 2.5(a). The titanium dioxide layer consists of two different regions: a region of stoichiometric TiO_2 and the other region consisting of oxygen deficient TiO_{2-x}. The TiO_{2-x} region is conductive due to the positively charged oxygen vacancies while the former region is less conductive. The overall structure can thus be regarded as a series combination of two resistors as shown in Figure 2.5(b), where the position of the barrier between the two regions determines the memristance. A positive voltage applied to the surface of TiO_{2-x} region repels the positively charged vacancies further into the TiO_2 region thereby moving the barrier further along and increasing the length of the conductive region. A negative voltage attracts the oxygen vacancies out of the TiO_2 region, thereby moving the barrier further back and reducing the length of the conductive region. When the highly resistive TiO_2 region occupies almost full length D, the device's equivalent resistance is R_{off}. Similarly, when the device is dominated by the highly conductive region, the equivalent resistance of the device is R_{on}.



(a) Physical structure of the memristor fabricated by HP Labs.



(b) Equivalent circuit model of HP Lab's memristor

Figure 2.5

Mathematically, the device can be modelled as;

$$v(t) = (R_{on}\frac{w(t)}{D} \ + \ R_{off}(1 - \frac{w(t)}{D}))i(t) \tag{2.11}$$

$$\frac{d(w(t))}{dt} \ = \ \mu_v \frac{R_{on}}{D^2} i(t) \tag{2.12}$$

where μ_v represents the average ion mobility.

Using (2.9) it is clear that the above device model represents a charge controlled memristor where:

$$x = w, \quad f(x,i) = \mu_v \frac{R_{on}}{D^2} i(t), \quad R(x) = R(w) = (R_{on}\frac{w(t)}{D} \ + \ R_{off}(1 - \frac{w(t)}{D})) \tag{2.13}$$

This initial model and derivation paved way for other more complex models that took into account the various effects encountered in memristors [16] such that more accurate simulations of these devices can be made to allow practical circuits to be built.

2.6.2 Other Models

For a memristor model to be termed effective, it has to meet the following criteria:

1. Accurate and computationally efficient.
2. Generic so as to fit different types of memristive applications.
3. Must work with thresholds (voltage and/or current) to enable accurate read and write.

Various models have been proposed in literature. Figure 2.6 shows a comparison of the models and Figure 2.7 shows their associated mathematical equation.

2.6.3 Window functions

The internal state variable w is bounded by 0 and the full thickness of the device D i.e the two boundary resistive states R_{off} and R_{on} respectively. To satisfy these bounds, (2.12) is multiplied by a function that nullifies the derivative, and forces (2.12) to be identical to zero when w is at a bound. One possible approach is an ideal rectangular window function (a function where the value is 1 for any value of the state variable, except at the boundaries where the value is 0). However, this doesn't take into account the nonlinear ion drift phenomenon, such as a decrease in the ion drift speed close to the bounds. Various window functions have been proposed in literature that take these non-idealities into account and are as summarised in Figure 2.8.

Model	Linear ion drift	Nonlinear ion drift	Simmons tunneling barrier	TEAM
State variable	$0 \leq w \leq D$ Doped region physical width	$0 \leq w \leq 1$ Doped region normalized width	$a_{off} \leq x \leq a_{on}$ Undoped region width	$x_{on} \leq x \leq x_{off}$ Undoped region width
Control mechanism	Current-controlled	voltage-controlled	Current-controlled	Current-controlled
Current-voltage relationship and memristance deduction	Explicit	I-V relationship: explicit Memristance deduction: ambiguous	Ambiguous	Explicit
Matching memristive system definition	Yes	No	No	Yes
Generic	No	No	No	Yes
Accuracy comparing practical memristors	Lowest accuracy	Low accuracy	Highest accuracy	Sufficient accuracy
Threshold exists	No	No	Practically exists	Yes

Figure 2.6: Comparison of different memristor models [17].

2.7 Memristors and Neuromorphic Engineering

The brain cortex is known to be marvellously efficient. A human brain, containing 10^{11} neurons and 10^{15} synapses, consumes 20W on average. Even a brain of a lower vertebrate excels the ultimate computer cluster ofthe time. A simulated mouse cortex on Blue Gene L based on Von-Neumann architecture, containing 8×10^4 virtual neurons and 5×10^{10} virtual synapses, consuming 40kW at the speed of 1GHz, is however still 10 times slower than a real mouse [18]. The fact is that the individual real neurons are as slow as 10Hz locally. It is the parallel architecture of computing and the fact that information storage and computation is done at the same time and in the same place that makes our brains a miracle.

In the pursuit for ever faster and more efficient processors a new interdisciplinary discipline that takes inspiration from biology, physics, mathematics, computer science and engineering to design artificial VLSI neural systems whose physical architecture and design principles are based on those of biological nervous systems was born.

A key aspect of neuromorphic design is understanding how information is transmitted and processed in neurons while incorporating learning. The remainder of this section will briefly review this and point out the connection to memristors.

2.7.1 Neurons and information processing in the brain

Neurons are specific cells comprising the major component of a biological cognitive system. Through a large range, signals in the form of spikes, known as the action potentials, propagate down the membrane of neural fibers in a network with a high interconnectivity to perform a style of parallel computation. Each neuron branches out two types of such fibers: axons and dendrites. Relatively long fibers branching out from a neuron, in a number of several hundred for each, are named as axons. Each axon makes connections with shorter fibers of other neurons, named as dendrites, which are in a much greater number. In fact, the contact

Model	Current-voltage relationship	State variable derivative
Linear ion drift	$v(t)=\left(R_{on}\frac{w(t)}{D}+R_{off}\left(1-\frac{w(t)}{D}\right)\right)i(t)$	$\frac{dw(t)}{dt}=\mu_v\frac{R_{ON}}{D}\cdot i(t)$
Nonlinear ion drift	$i(t)=w(t)^n\beta\sinh(\alpha\, v(t))+\chi[\exp(\gamma\, v(t))-1]$	$\frac{dw(t)}{dt}=a\cdot f(w)\cdot v(t)^m$
Simmons tunneling barrier	$v(t)=\left[R_{on}+\frac{R_{OFF}-R_{ON}}{x_{off}-x_{on}}(x-x_{on})\right]i(t)$ or $v(t)=R_{ON}\cdot e^{\frac{\lambda}{x_{off}-x_{on}}(x-x_{on})}\cdot i(t)$ Note that this is different than original Simmons tunneling barrier	$\frac{dx(t)}{dt}=\begin{cases}C_{off}\sinh\left(\frac{i}{i_{off}}\right)\exp\left[-\exp\left(\frac{x-a_{off}}{w_c}-\frac{\lvert i\rvert}{b}\right)-\frac{x}{w_c}\right], & i>0\\ C_{on}\sinh\left(\frac{i}{i_{on}}\right)\exp\left[-\exp\left(\frac{x-a_{on}}{w_c}-\frac{\lvert i\rvert}{b}\right)-\frac{x}{w_c}\right], & i<0\end{cases}$
TEAM	$v(t)=\left[R_{on}+\frac{R_{OFF}-R_{ON}}{x_{off}-x_{on}}(x-x_{on})\right]i(t)$ or $v(t)=R_{ON}\cdot e^{\frac{\lambda}{x_{off}-x_{on}}(x-x_{on})}\cdot i(t)$	$\frac{dx(t)}{dt}=\begin{cases}k_{off}\cdot\left(\frac{i(t)}{i_{off}}-1\right)^{\alpha_{off}}\cdot f_{off}(x), & 0<i_{off}<i\\ k_{on}\cdot\left(\frac{i(t)}{i_{on}}-1\right)^{\alpha_{on}}\cdot f_{on}(x), & i<i_{on}<0\\ 0, & otherwise\end{cases}$

Figure 2.7: The mathematical equation used in each of the memristor models [17].

between an axon and a dendrite is separated with a tiny cleft of around 20nm, known as the synapse. This is illustrated in Figure 2.9. Figure 2.10 illustrates two neurons connected by a synapse. The pre-synaptic neuron is sending a pre-synaptic spike $V_{mem-pre}$(t) through one of its axons to the synaptic junction. Neural spikes are membrane voltages from the outside of the cellular membrane V_{pre+} with respect to the inside V_{pre-} . Thus $V_{mem-pre} = V_{pre+} - V_{pre-}$ and $V_{mem-pos} = V_{pos+} - V_{pos-}$. The "large" membrane voltages during a spike (in the order of a hundred mV) cause a variety of selective molecular membrane channels to open and close allowing many ionic and molecular substances to flow, or preventing them from flowing through the membrane. At the same time, synaptic vesicles inside the pre-synaptic cell containing "packages" of neurotransmitters fuse with the membrane in such a way that these "packages" are released into the synaptic cleft (the inter cellular space between both neurons at the synaptic junction). Neurotransmitters are collected in part by the post-synaptic membrane, contributing to a change in its membrane conductivity. The cumulative effect of pre-synaptic spikes (coming from this or other pre-synaptic neurons) will eventually trigger the generation of a new spike at the post-synaptic neuron. Each synapse is characterized by a "synaptic strength" (or weight) w which determines the efficacy of a pre-synaptic spike in contributing to this cumulative action at the post-synaptic neuron. In other words, acts as an information filter. This weight w can be interpreted as the size and/or number of neurotransmitter packages released during a pre-synaptic spike

The synaptic weight w is considered to be non-volatile and analog in nature, but it changes in time as a function of the spiking activity of pre- and post-synaptic neurons. This phenomenon

Function	Joglekar	Biolek	Prodromakis	TEAM
$\frac{f(x)}{f(w)}$	$f(w)= 1-(2\frac{w}{D}- 1)^{2p}$	$f(w)= 1-(\frac{w}{D}- stp(-i))^{2p}$	$f(w)= j(1-[(w- 0.5)^2+ 0.75]^p)$	$f_{on,off}= \exp[-\exp(\frac{\lvert x- x_{on,off}\rvert}{w_c})]$
Symmetric	Yes	Yes	Yes	Not necessarily
Resolve boundary conditions	No	Discontinuities	Practically yes	Practically yes
Impose nonlinear drift	Partially	Partially	Partially	Yes
Scalable $f_{max}< 1$	No	No	Yes	No
Fits memristor model	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	TEAM for Simmons tunneling barrier fitting

Figure 2.8: Comparison of different window functions [17].

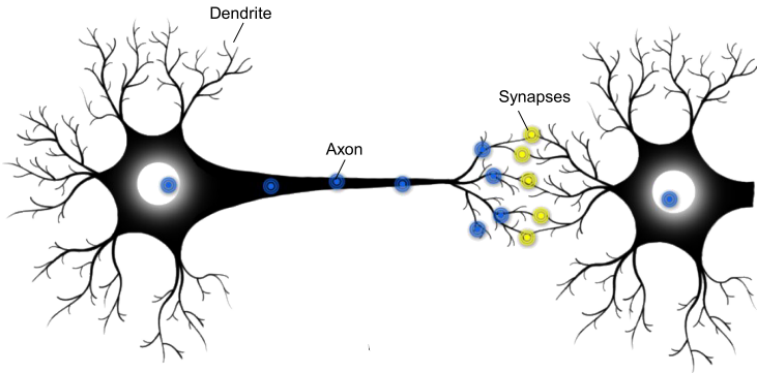


Figure 2.9: An illustration of two connected neurons.

was originally observed and reported in 1949 by Hebb, who introduced his Hebbian learning postulate [20]: *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."* Traditionally, this has been described by computational neuroscientists and machine learning computer engineers as producing an increment in synaptic weight Δw proportional to the product of the mean firing rates of pre- and post-synaptic neurons. Biologically, this change is known as plasticity and has experimentally been shown [21] using sea snail (Aplysia Californica) to be the origin of learning.

Note: STDP (Spike-timing-Dependent-Plasticity) a refinement of hebbian learning (takes into account the precise relative timing of individual pre and post-synaptic spikes rather than their average rates over time) will not be considered in this project.

2.7.2 Memristors are synapses

Other than having non-volatile continuous analog memristive states that can represent the synaptic weight, memristors also demonstrate both long and short term plasticity making them

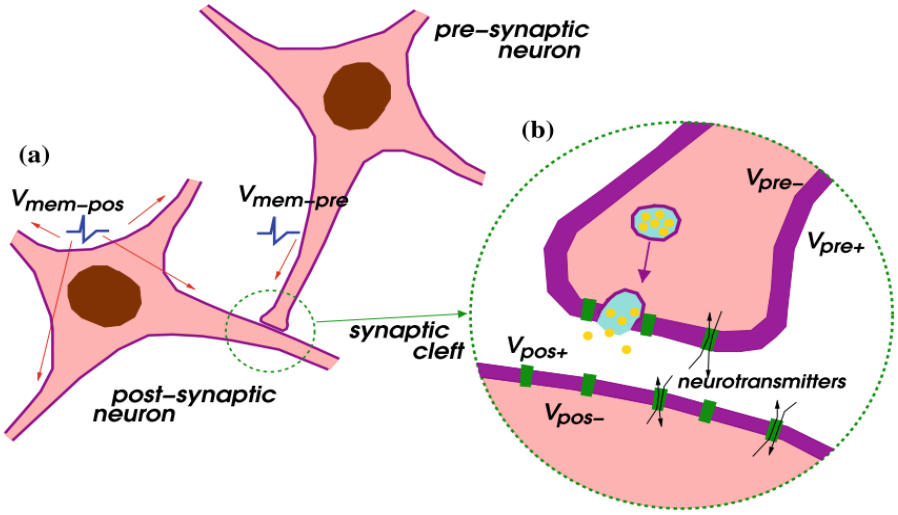


Figure 2.10: Illustration of synaptic action [19]. (a) The pre-synaptic neuron sends an action potential $V_{mem-pre}$ traveling through one of its axons to the synapse. The cumulative effect of many pre-synaptic action potentials, generates a post-synaptic action potential at the membrane of the post-synaptic neuron, which propagates through all the neuron's terminations. (b) Detail of synaptic junction. The cell membrane has many membrane channels of varying nature which open and close with changes in the membrane voltage. During a pre-synaptic action potential vesicles containing neurotransmitters are released into the synaptic cleft

the perfect models for synapses. [23] describes an experiment performed on Hippocampus neurons found in the brain that were stimulated by a brief high frequency pulse train producing a long-lasting enhancement (long-term plasticity) in the strength of the stimulated synapses compared to a single pulse. An overview of the experiment is presented in Figure 2.11. This response is exactly whats observed in memristors and is the mechanism behind which their state can be read without modification and changed (written to), as seen in later sections.

2.7.3 Neural Networks

Artificial neural networks (ANNs) commonly referred to as just Neural Networks mimic the structure of the brain to try and achieve the same processing capabilities that can model complex systems which cannot be otherwise modelled mathematically. The processing elements in an ANN is analogous to the neuron in that they have many inputs (dendrites) and perform a multiply accumulate (MAC) function on the inputs as shown in Figure 2.12. This result is then subjected to a nonlinear filter usually called a transfer function, which is usually a threshold function or a bias in which output signals are generated only if the output exceeds the threshold value. Alternately, the output can be a continuous function (typically a sigmoid function limited to the range 0 to +1 or a hyperbolic tangent function limited to -1 to +1) of the combined input. The output of a processing element (axon) branches out and becomes the input to many other processing elements. These signals pass through connection weights (synaptic junctions) that correspond to the synaptic strength of the neural connections. The input signals to a processing element are modified by the connection weights prior to being summed by the processing element. These processing elements are usually organized into a sequence of layers with full or random connections between layers. Typically, there are three or more layers: an input layer where data is presented to the network through an input buffer, an output layer with a buffer that

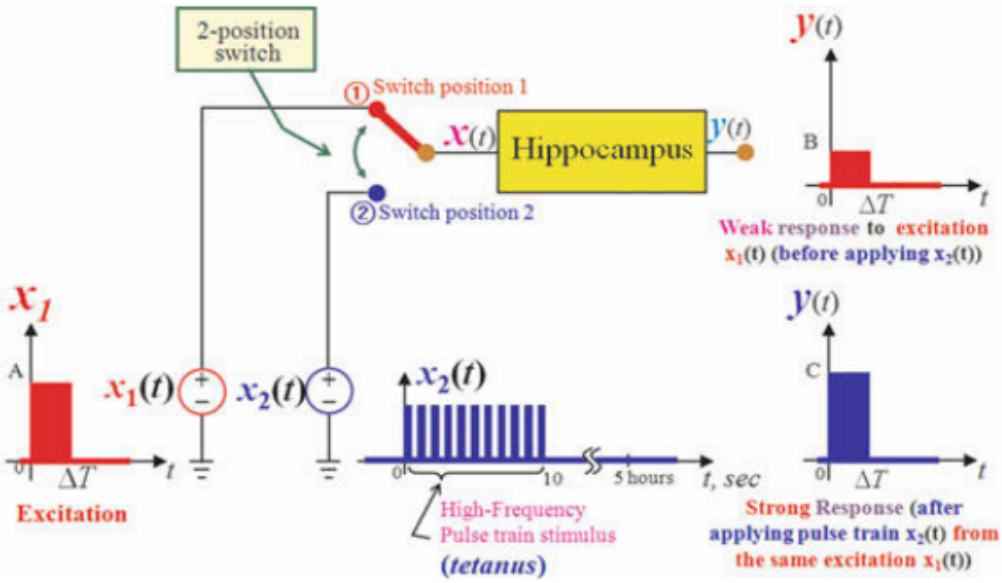


Figure 2.11: Simplified schematic diagram depicting the classic Bliss-Lomo experimental set-up [23].

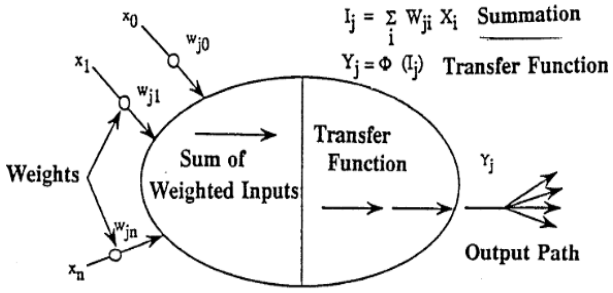


Figure 2.12: Schematic Representation of an Artificial Neuron [24].

holds the output response to a given input, and one or more intermediate or "hidden" layers. A typical neural network arrangement is shown in Figure 2.13.

The operation of an ANN involves two main stages: training/learning and recall/inference. Learning is the process of adapting the connection weights in response to stimuli presented at the input buffer. The network "learns" in accordance with a learning rule which governs how the connection weights are adjusted in response to a learning example applied at the input buffers. Recall is the process of accepting an input and producing a response determined by the trained network.

There are several different kinds of learning commonly used with neural networks. Perhaps the most common is the so-called supervised learning in which a stimulus is presented at the input buffer of the network and the output from the output buffer is sent to a system that compares it with a desired output and then uses a corrective or learning algorithm (commonly back-propagation) to convert the difference (error signal) into an adjustment of the weighting coefficients (connection weights) that control the inputs to the various processing elements.

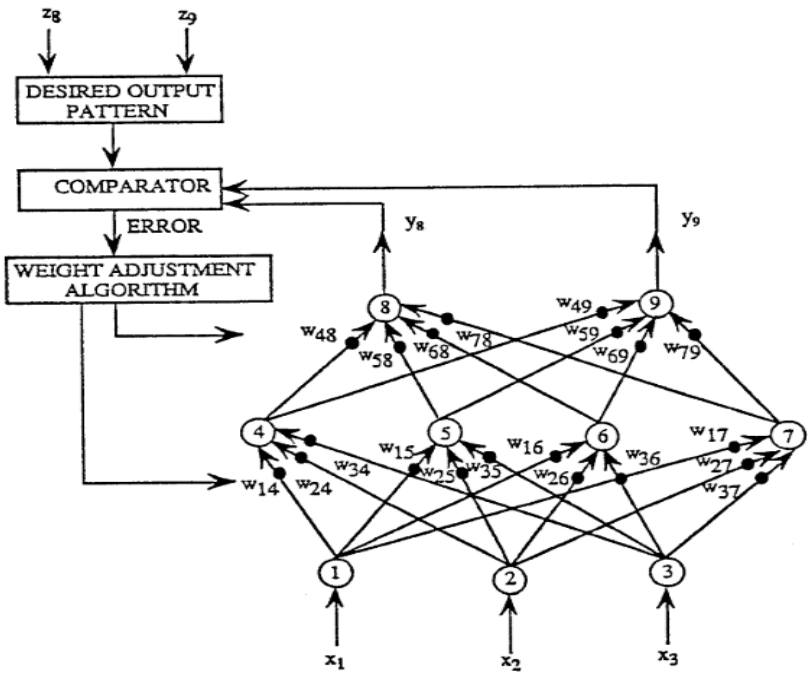


Figure 2.13: An Artificial Neural Network with Supervised Learning [24]

2.8 Crossbars

To realize the benefits of memristor devices ideally the ANN will be implemented in a crossbar array architecture which offers high density, random access and large connectivity, as shown in Figure 2.14. In addition, the configuration can be fabricated over underlying circuity to save space. The network can be mapped to this architecture by connecting an input neuron at each top electrode (horizontal lines in Figure 2.14) and an output neuron at each bottom electrode (vertical lines in Figure 2.14). In this configuration, each input neuron is individually connected to every output neuron, with each connection implemented through a memristor which acts as a synapse with a particular weight that is represented by its conductance. The neural resemblance of a crossbar is profound since each output neuron can perform a MAC operation $I_j = \sum_{i=1}^{N} V_i \cdot G_{ij}$ (where z is the total current flowing through the vertical (bit) line, x is the voltage applied on a horizontal (word) line and w is the conductance G of a memristor) on all its input neurons without any additional hardware.

2.9 Sneak Paths

As good as the crossbar architecture is, it has a major limitation with the use of memristor; it is prone to current leakages (sneak-paths). Sneak-paths are undesired current paths through neighbouring cells when a particular cell in the crossbar is selected for a read or write operation by applying for a voltage on the selected memristors wordline and connecting the bitline to a current sensing scheme as shown in Figure 2.15(a). The resistance of unselected memristors combine to form a parallel resistance path with the resistance of the desired memristor as illustrated by the equivalent circuit in Figure 2.15(b). I_{sneak} (the current through the unselected memristors) is data-dependent as well as array size dependent. A larger array with more low resistance unselected cells will result in high I_{sneak} value. Sneak-paths particularly lead to erroneous reading of a memristance because of interference of undesired currents from the

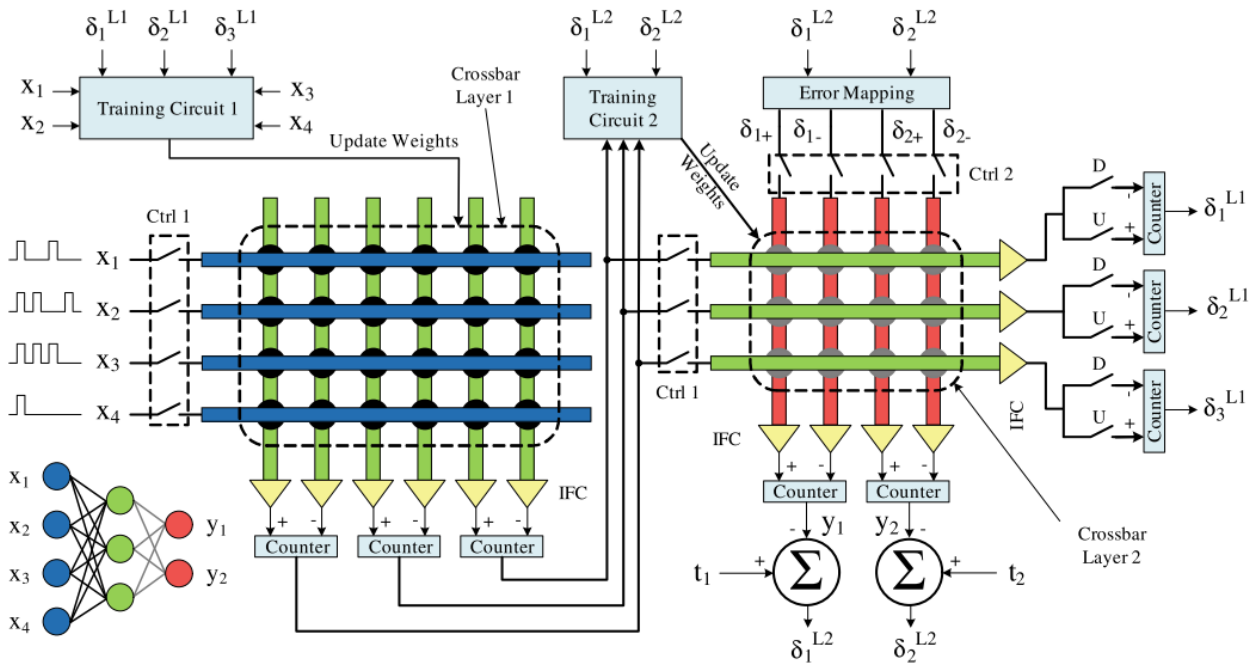


Figure 2.14: Hardware implementation of a two layer memristor crossbar array implemented in [33]. A 4×3×2 neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure.

unselected neighbouring cells and during write operation, sneak-paths can cause unselected memristors to unintentionally change their memristance.

Moreover, Sneak-currents result to high power consumption, limitation on the maximum array size, decline in the effective read margin among others.

2.9.1 Sneak-path Elimination Techniques

Various methods have been proposed for addressing the sneak-paths problem in crossbar architectures [12]. Each of these methods have their pros and cons and there is not a generally accepted solution to this problem without tradeoff of one or more of the performance metrics. This section presents a review of some of these methods. The existing sneak-path elimination methods can be subdivided into three categories as explained in the following section and Table 3.1 presents a summary of all the discussed techniques.

- **Additional Crosspoint Device:** In this technique, sneak-paths are prevented by restricting current access to other memristors other than the one selected. An isolating device alongside the memristor at every crosspoint of the crossbar is used. Several options exist, a one transistor and one memristor (1T1M) structure, one diode and one memristor (1D1M), and one memistor and one memristor (1M1M). These techniques are able to minimise sneak-path in the array but at the expense of area per memristor, array density, 3-D integration, fabrication issues as well as power inefficiency among other shortcomings.

- **Complementary Resistive Switch:** Rather than combining a memristor with an isolating device this method instead combines two memristor cells together anti-serially

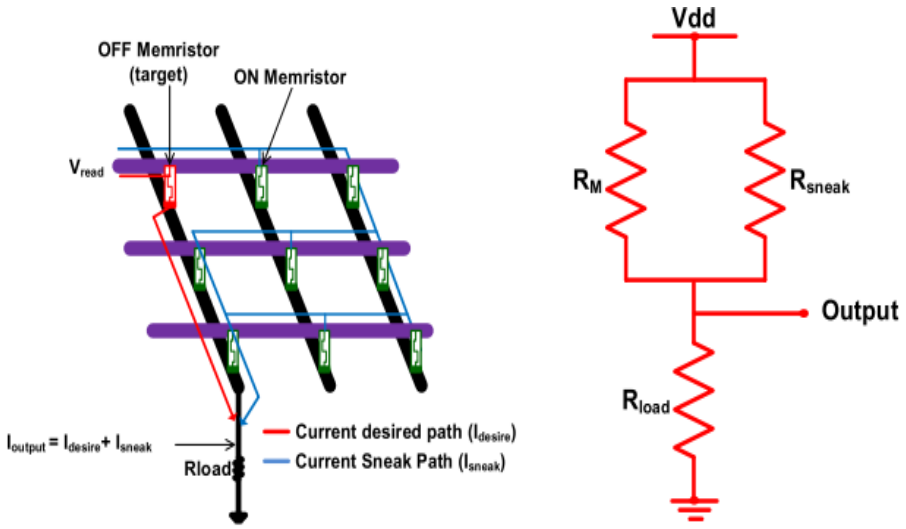


Figure 2.15: Demonstration of sneak-paths effect in crossbar architecture (a) Reading from a crossbar Array. I_{desire} is the current of the desired cell (Red cell) while I_{sneak} is the sneak-path current from undesired cells, both currents combine to form I_{out} put which is not the desired output (b) Equivalent circuit model of sneak-paths effect.[12]

into a CRS. The logic states of the CRS result in high resistance values reducing sneak-paths through the neighbouring cells because of their high resistances.

- **Multi-step Reading:** Multi-step reading involves executing the read operation over a minimum of two or more stages before the correct memristance value can be correctly determined. For example, in a two-step read operation the first step senses the leakage current, the second step senses the overall current and the difference between the output of both stages represent the current from the desired memristor.

- **Line Biasing:** Sneak-path can also be minimised via the voltages applied to the word and bit lines. [34] discussed four different techniques in this category: Floating rows and columns (FRFC) grounded columns and rows (GRGC), grounded columns and floating rows (FRGC) and floating columns and grounded rows (GRFC). These biasing techniques could however not ensure a sufficient read margin beyond an array size of 64×64. GRGC, GCFR and FCGR also consumes excess power because of their connection to the ground.

Note: A key aspect of this project involves investigating how well ANNs can perform in the presence of sneak-paths, thus additional crosspoint devices or CRS which eliminate sneak-paths will not be considered when designing the system.

In an ANN, three operations can take place on the memristors - Reading, Writing and Inference. The first two operations take place on one or several memristors while the last operation takes place on all the memristors. In [12] it was analytically shown using bipolar memristors that sneak paths cannot be eliminated when reading memristors with different weights but can be minimised by selecting more cells (along a wordline) for read. Furthermore, the following line biasing techniques which can mitigate sneak currents that do occur during a write operation were proposed.

2.10 Line biasing techniques for crossbar array write operation

Figure 2.16 classifies the different memristors during a write operation and presents an equivalent circuit model of the crossbar. Note: when the term 'unselected' is used it represents R_m, R_n and R_{mn} unless otherwise stated.

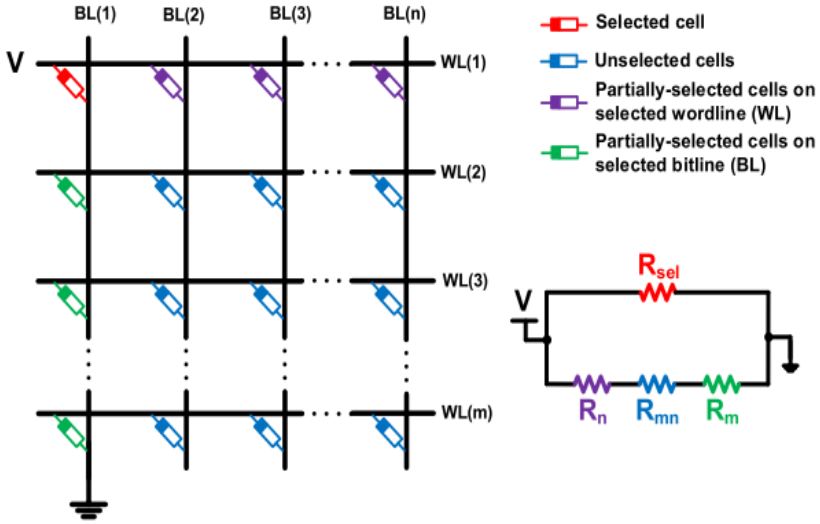


Figure 2.16: An m×n memristor-based crossbar memory structure set-up for write operation and its equivalent circuit model that depicts the effect of all resistances in the array during a write operation. R_{sel} represents the selected cell $R_{1,1}$. R_m and R_n represent the row and column partially-selected cells respectively and R_{mn} simplify the cells in the unselected lines

2.10.1 Floating Lines Write Scheme

Figure 2.16 is a representation of this scheme. The selected cell in this scheme usually switches successfully, however, the state of the partially-selected cells on the selected wordline (purple) and bitline (green) might not be preserved. The voltages on the unselected lines, V_{word} and V_{bit} are dependent on the values of m and n hence $V_{word} = V_{bit} = f(m, n)$. [12] derives an analytic expression for V_m, V_n, V_{mn} and V_{sel} based on the equivalent circuit and evaluates them to plot the graph in Figure 2.17.

The voltage drop across the selected cell is always V_w. For partially-selected cells the voltage drop doesn't exceed $V_w/2$ only if m = n (square crossbar) but exceeds it and approaches V_w if m≫n or n≫m. On the contrary, unselected (blue) cells receive less voltage as the array size increases irrespective of the array shape. Thus, the success of the floating scheme depends heavily on the size and aspect ratio of the crossbar array.

2.10.2 V/2 Write Scheme

This scheme is commonly used as the choice for write operation in crossbar arrays. The structure of the V/2 scheme is shown in Figure 2.18. A voltage of V_w and zero are applied to the word line and the bit line of the target memristor respectively and all other lines are biased with a voltage

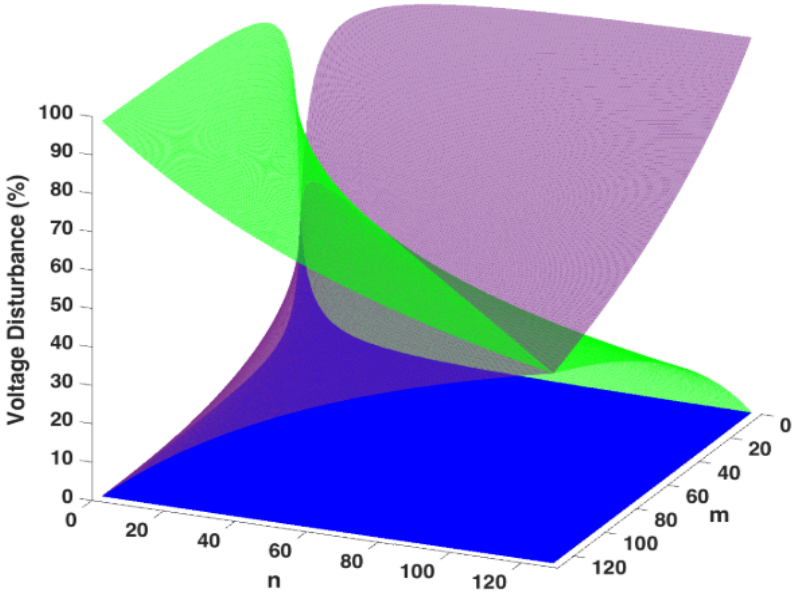


Figure 2.17: Voltage drop on unselected cells as array size varies when unselected lines are left floating. Voltage on partially-selected cells are depicted in purple and green, while unselected cells are represented in blue [12].

of $V_w/2$. The voltage bias helps to minimise current leakages to some of the unselected cells. In this scheme, the majority of the unselected cells are totally protected against disturbance. To be specific, $((m-1)(n-1))$ cells are protected, these are the unselected memristors. The remaining $m+n-2$ memristors (partially-selected ones) in the m×n array are exposed to a voltage drop of Vw/2, which ideally shouldn't change their state. The number of cells $(m+n-2)$ susceptible to voltage disturbance in this scheme is less than 50% of the total cells in the array (mn) (percentage reduces as the array size grows). However, the probability of write error is slightly high in the V/2 scheme because the partially-selected cells $(m+n-2)$ have a voltage of $V_w/2$ across them constantly resulting to their state being perturbed if the voltage is applied long enough.

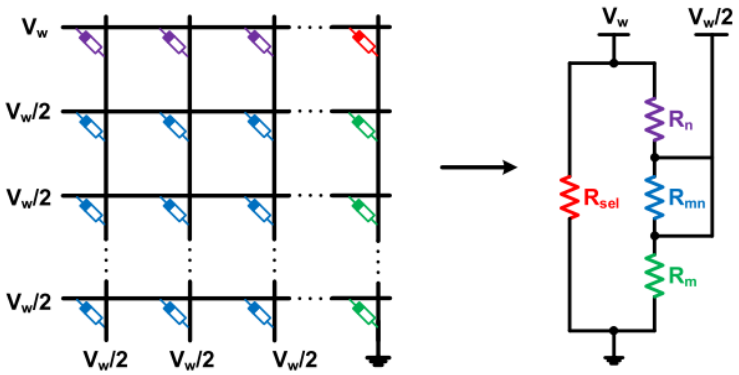


Figure 2.18: Structure of V/2 write scheme and its equivalent circuit level model. The selected cell in red has a voltage of V_w across it while the purple and green partially-selected cells have a voltage drop of $V_w/2$ on them. The unselected blue cells have zero voltage across them.

2.10.3 V/3 Write Scheme

This scheme is similar to the previous scheme however the unselected cells are biased with a voltage of $V_w/3$ and $2V_w/3$ applied to their wordlines and bitlines respectively as shown in Figure 2.19. Since these cells are biased with a smealler voltage the probability of their state being disturbed is low. However, more cells (mn - 1) are affected by sneak-paths.

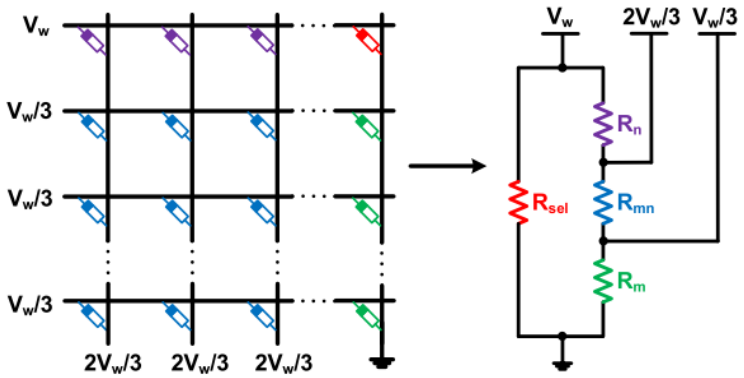


Figure 2.19: Structure of V/3 write scheme and its equivalent circuit level model. The selected cell in red has a voltage of Vw across it while the other cells (purple, green and blue) has a voltage drop of $V_w/3$.

2.10.4 Summary and power consumption

All the three aforementioned write schemes have their pros and cons without the effect of line resistance. The floating scheme is only considered reliable when the array is square shaped; otherwise state of unselected cells are perturbed. The V/2 scheme protects more cells than any other scheme but the state of the m+n-2 cells constantly exposed to a voltage of $V_w/2$ can be overwritten if the write voltage is applied for a longer time. The V/3 scheme on the other hand, exposes all the groups of unselected cells to a safe low voltage of Vw/3. Figure 2.20 summarises and compares the different techniques. Lastly, the power consumption of these schemes will be

	Voltage on selected cell(s)	Voltage on Unselected Cells		Total Current Leakage	No. of disturbed cells	Dependence on Array Size/ Aspect Ratio	Probability of unselected cells Switching
		Min. Voltage	Max. Voltage				
Floating Lines	V_w	$\frac{V_w}{m+n-1}$	$\max\left(\frac{V_w(m-1)}{m+n-1}, \frac{V_w(n-1)}{m+n-1}\right)$	$\frac{V_w(m-1)(n-1)}{R(m+n-1)}$	$mn-1$	Yes	≤ 0.5 (if $m=n$)
V/2	V_w	0	$V_w/2$	$\frac{V_w(m+n-2)}{2R}$	$m+n-2$	No	0.5
V/3	V_w	$V_w/3$	$V_w/3$	$\frac{V_w(mn-1)}{3R}$	$mn-1$	No	0.33

Figure 2.20: Comparative summary of write schemes' performances without the effect of line resistance [12].

analysed, [12] derives the analytic expressions to work them out. Figure 2.21 shows an ideal

power consumption comparison between the three schemes in the worst case scenario. The V/3 scheme which is the most reliable write scheme consumes enormous power. The huge power consumption of the V/3 scheme is as a result of frequent switching between the Vw/3 and 2Vw/3 power rail as well as the fact that all the cells in the array experience some degree of sneak-path current. Figure 2.22 shows the power consumptions of the floating lines scheme compared against the V/2 write scheme so as to show cases where the dimensions of the array are nonsquare (m≠n). As m deviates from n, there is more power savings with the floating lines than the V/2 scheme but the inequality between m and n causes further disturbance to the unselected cells which could change their state.

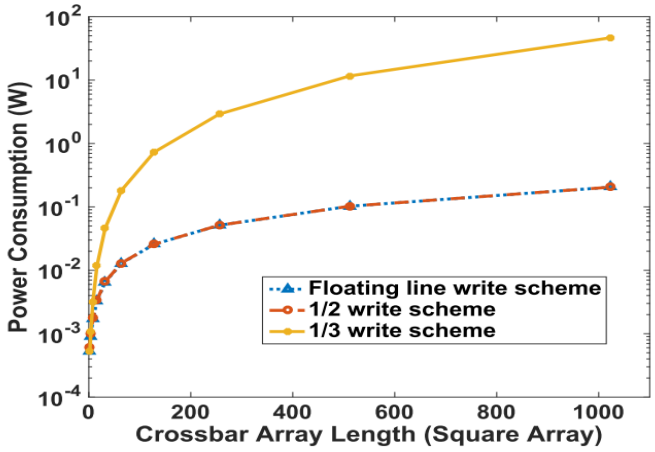


Figure 2.21: Comparison of power consumptions in the three write schemes in the worst case scenario (all unselected cells are initialised to Ron). R_{on} = 10kΩ, R_{off} = 100KΩ, R_{off}/R_{on} = 101. Power value is scaled up equally for the three schemes in order to show the difference between the schemes [12].

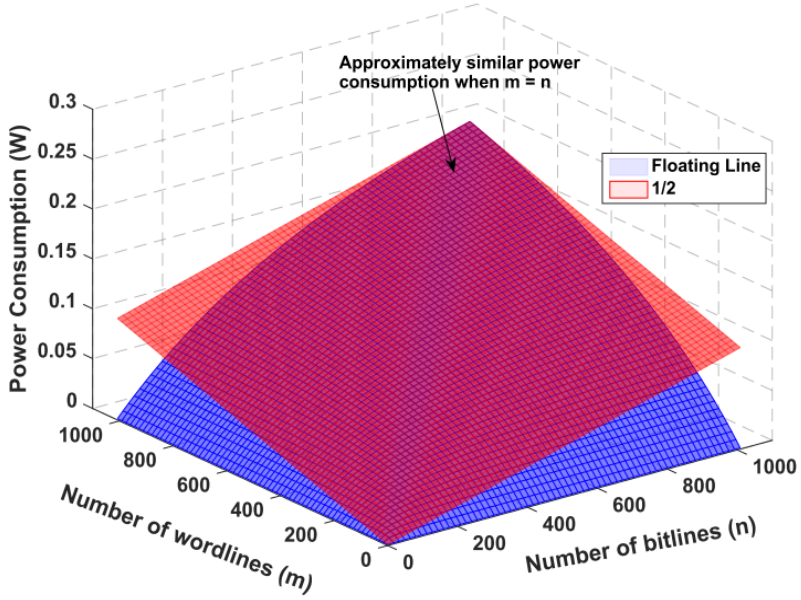


Figure 2.22: Comparison of power consumptions between the floating lines and V/2 write schemes over varying range of array sizes and structures. Power value is scaled up equally for the two schemes in order to show the difference between the schemes [12].

Chapter 3

Preliminary Implementation

This section will detail the work that was conducted in order to get a better understanding of the behaviour of memristors, how memristors can be programmed and to lay the foundation to implementing a NN using crossbar architectures.

3.1 Selecting an appropriate memristor model

The memristor model selected in this project is based on [35], where the device current–voltage characteristics and resistive switching rate are expressed as a function of: 1) bias voltage and 2) initial resistive state. The model's window function is voltage dependent rather than having constant resistive boundaries of operation. The differential algebraic equation (DAE) set is presented below:

$$i(R,v) = \begin{cases} a_p(1/R)sinh(b_p v), & v > 0 \\ a_n(1/R)sinh(b_n v), & v < 0 \end{cases} \tag{3.1}$$

$$\frac{dR}{dT} = g(R,v) = s(v) * f(R,v) \tag{3.2}$$

where *s(v)* is the switching function and *f(R,v)* the window function as defined below

$$s(v) = \begin{cases} A_p(-1 + e^{t_p|v|}), & v > 0 \\ A_n(-1 + e^{t_n|v|}), & v < 0 \\ else, & 0. \end{cases} \tag{3.3}$$

$$f(R,v) = \begin{cases} -1 + e^{\eta k_p(r_p(v)-R)}, & R < \eta \cdot r_p(v) \\ -1 + e^{\eta k_n(r_n(v)-R)}, & R < \eta \cdot r_n(v) \\ else, & 0. \end{cases} \tag{3.4}$$

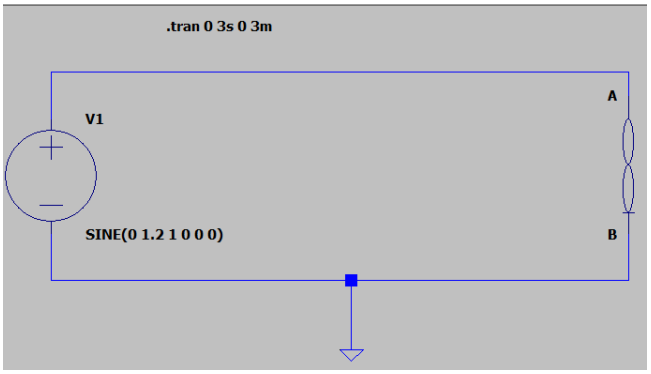
Symbols $A_{p,n}$, $b_{p,n}$, $A_{p,n}$, $t_{p,n}$, and $k_{p,n}$ are parameters that fit the model on an arbitrary memristor. To obtain them, a proper device-characterization procedure complemented by a parameter extraction algorithm was carried out as shown in [35]. $r_{p,n}(v)$ are the voltage dependent resistive boundary functions for the positive and negative stimulation cases. For any RS below $r_p(V)$, at a given positive voltage V, applying this voltage can push the device to $r_p(V)$, but not further (saturation). If the device is already above $r_p(V)$, no switching occurs (for positive stimulation). The same applies for negative biasing and the $r_n(V)$ limit.

The DAE set can be converted to RS time-response equations analytically under constant bias voltage as shown below:

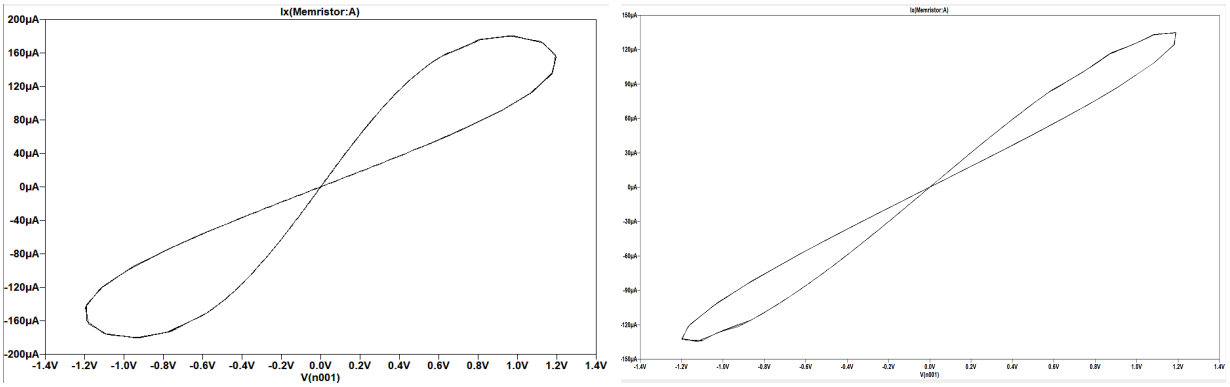
$$R(t)|_{V_b} = \begin{cases} \frac{ln(e^{\eta k_p r_p(v)} + e^{-\eta k_p s_p(v) t} * (e^{\eta k_p R_o} - e^{\eta k_p r_p(V_b)})}{K_p}, & V > 0 \quad R < \eta r_p(v) \\ \frac{ln(e^{-\eta k_n R_o + \eta k_p s_p(v) t}) - e^{\eta k_n r_n(V_b)}) * (-1 + e^{\eta k_n s_n(v) t})}{k_n}, & V < 0 \quad R > \eta r_n(v) \\ R_0, & otherwise \end{cases} \tag{3.5}$$

The model was compared to other generalized, well established memristor models published, one of which is the VTEAM model described in Section 2.6.2 and it was found to outperform it on various memristor characterization data based on % RMS error. Moreover, the model captures its memristive effect without requiring integration of the model state variable, making it suitable for fast and/or large-scale simulations and overall inter-operable with current design tools. The model can be found in Appendix A.

The model is only valid for pulsed inputs, to produce the characteristic hysterisis loop of a memristor a different model based on the linear ion drift model and Joglekar window function is used (using LTSpice) to produce Figure 3.1. The authors of the selected model (henceforth



(a) Schematic showing sinusoidal input to memristor



(b) At 1Hz (c) At 1.5Hz

Figure 3.1: Characteristic hysteresis loops, loop area shrinks with increasing frequency.

referred to as 'Soton' model) published an IEEE letter [38] to provide examples of their working Memristor model. It was shown that memristance of the models increase with an increasing number of pulses with larger amplitude pulses converging to R_{Max} faster. This was verified in Figure 3.2 and Figure 3.3 shows the schematic implementing it (produced using Cadence

Virtuoso). Pulses of larger pulse widths have a similar relation to pulses of increasing amplitude and hence produce similar graphs.

Note: In order to extract the resisitive state (RS), the RS was output from the model as a voltage and hence the y-axis units are Volts rather than Ohms.

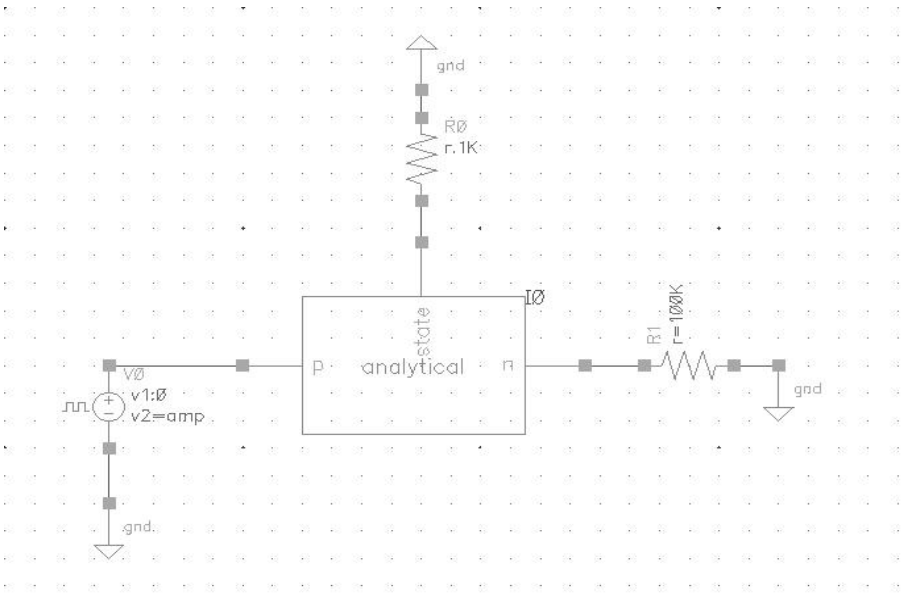


Figure 3.2: Schematic

3.2 Reading and writing to a memristor

When reading the state of a memristor it is important that a read pulse does not perturb its current state. From, Eqn 3.2 it can be noted that the state change is an exponential function and hence using small pulsed amplitudes reduces unwanted state perturbations. This is demonstrated in Figure 3.4 where a series of pulses was applied to the memristor, followed by a brief pause and then another read pulse. A zoomed in version of the final write pulse and read pulse can be seen in Figure 3.5, where it can be observed that a read pulse of a small amplitude only changes the RS (marked on the figure) by around 1Ω.

Most existing Memristors (uncited although from reliable source) exist within the range of 5K-30K Ω. The Soton model yielded results of low RMS Error for their fabricated memristors within the range 10K-17K Ω and it is hoped that this extends for a larger memristance range as shown in Figure 3.6. In addition, from the figure it is possible to work out the number of pulses needed to set the memristor to a particular RS given the pulse amplitude.

3.3 Reading and writing to memristors in a crossbar

The 7x7 memristor crossbar in Figure 3.7 will aid in demonstrating the read and write capabilities of memristors in a crossbar. Besides the crossbar, Figure 3.7 consists of a Verilog-A module that passes read and write pulses to the word line and a current sensing mechanism on each bit

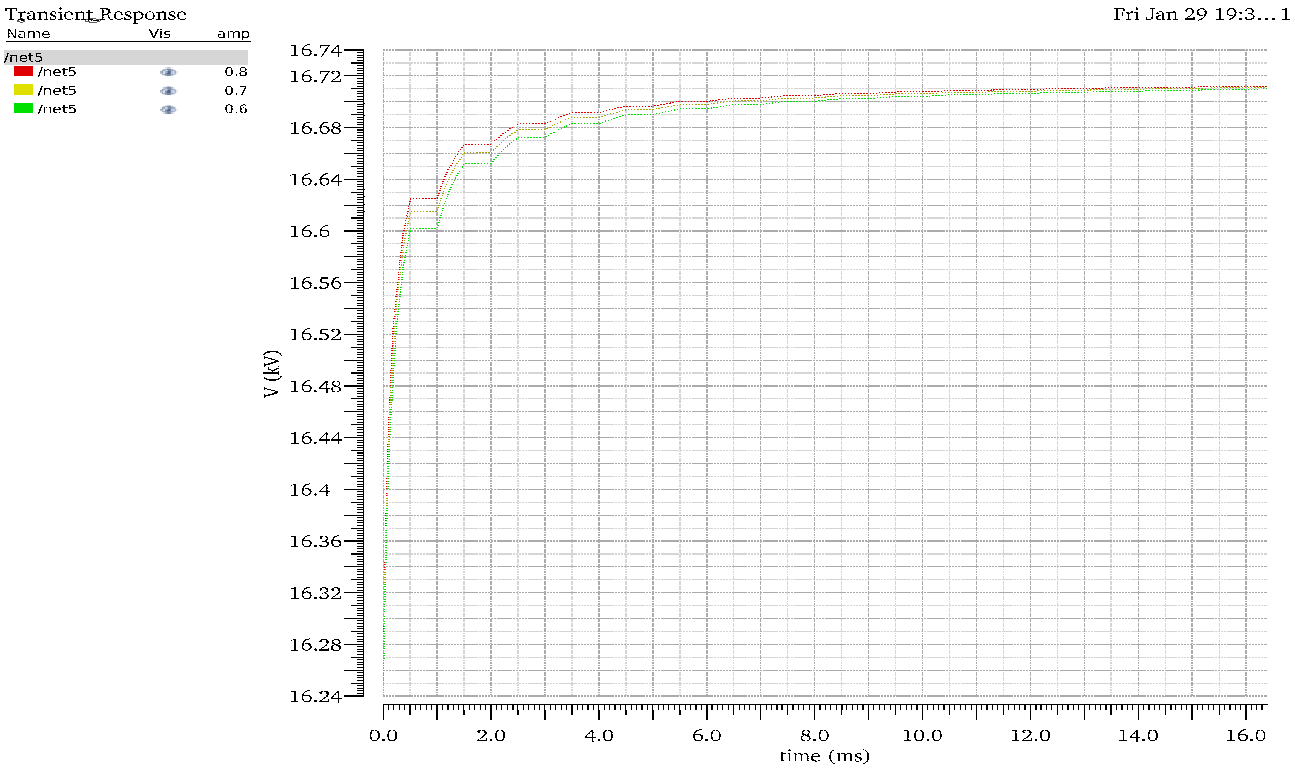


Figure 3.3: Memristance change for pulses of different amplitude

line that biases the crossbar according to the V/2 write scheme.

Several current sensing mechanisms have been proposed in literature: [36] proposed using a switch capacitor subtractor as shown in Figure 3.8, [37] proposed a voltage divider or using a transimpedance amplifier (TIA) based read circuit as shown in Figure 3.9 and lastly, a Sigma-Delta read circuit was also proposed as shown in Figure 3.10. All of the circuits are viable choices. However, as the project is focused on understanding the behaviour of memristors an ideal transimpedance amplifier based on a VCVS (voltage-controlled voltage source) was selected to eliminate the effect of non-linearities on the ANN performance. Figure 3.11 shows a testbench used to demonstrate the behaviour of the VCVS and the graph in Figure 3.12 shows the ideal current to voltage conversion.

The graph in Figure 3.13 demonstrates a write operation followed by a read operation on the memristor located on the 4^{th} wordline and 4^{th} bitline (4,4). Initially, during the write operation, pulses of 700mV (Vw) are passed to the target wordline and the NMOS on the target bitline is turned on to ground the line. This modulates the target memristors state to around 20KΩ. During a read operation, a pulse of 700mV is passed to the target wordline while, the PMOS that connects the target bitline to the TIA is turned on to allow a reading to be taken. The target memristor has a voltage of only 350mV (Vw/2) thus, its state can be read without pertubation. From the current on the bitline and hence voltage output of the TIA the RS of the target memristor can be derived using Ohms Law.

Using this method, it is possible to read the states of all the memristors on the same wordline confidently. Moreover, without significantly pertubing the state of other memristors, it is possible write to all the memristors on the same wordline provided they all need to be set to

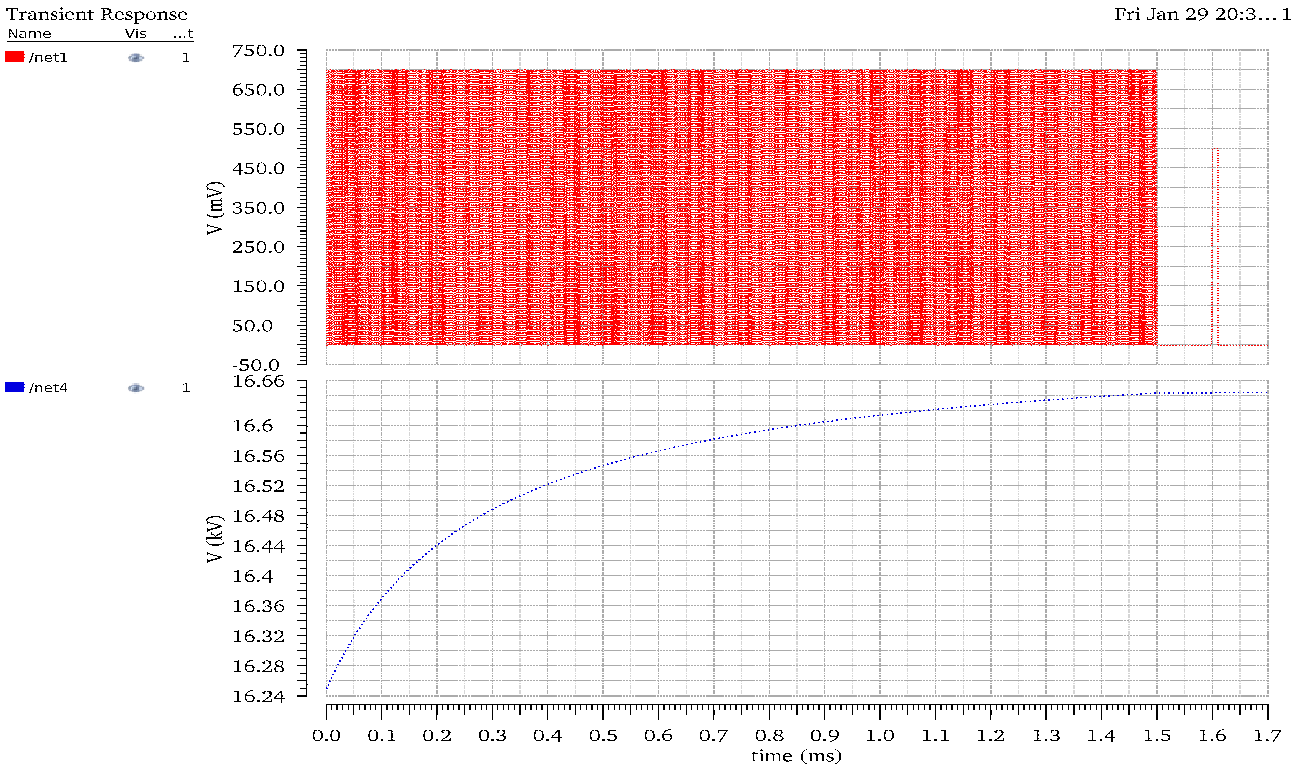


Figure 3.4: Writing to a memristor and reading its state

the same RS.

3.4 Algorithmic mitigation of sneak paths

The effect of sneak currents in selectorless crossbars have deterred their application in industry. If a technique existed to eliminate this effect, it could eliminate the need of crossbars with a selector device per memristor and result to a significant number of new applications in industry. Thus, as a matter of interest, this section was pursued.

As mentioned in Section 2.18, the partially selected memristors will experience a voltage of Vw/2 and thus their RS will be perturbed by sneak currents while the unselected memristors will experience minimal sneak currents. Thus, my initial intuition was to try mitigate the effect on the partially selected memristors by passing pulses of opposite polarity. Figure 3.14 shows a simple 3x3 selectorless memristor crossbar array that was used to initially test this algorithm and Figure 3.15 shows a graph that tracks the RS of various labelled memristors and the voltages applied on certain wordlines.

Referring to the Verilog-A code in Appendix C, initially all the memristors are set to 12.5KΩ, then the target memristor is set to around 16.64KΩ during which wordlines 2 and 3 are held at Vw/2 and bitlines 1 and 2 to try mitigate sneak currents flowing into the unselected memristors. Then, a series of negative pulses are applied first on wordline 2 and then 3 to try bring back the RS of the partially memristors on the bitline back to to 12.5KΩ. This can be seen from the RS tracks in Figure 3.15. A negative pulse cannot be applied on the target wordline as this would perturb the state of the target memristor. Therefore, this algorthm was partially successful; the

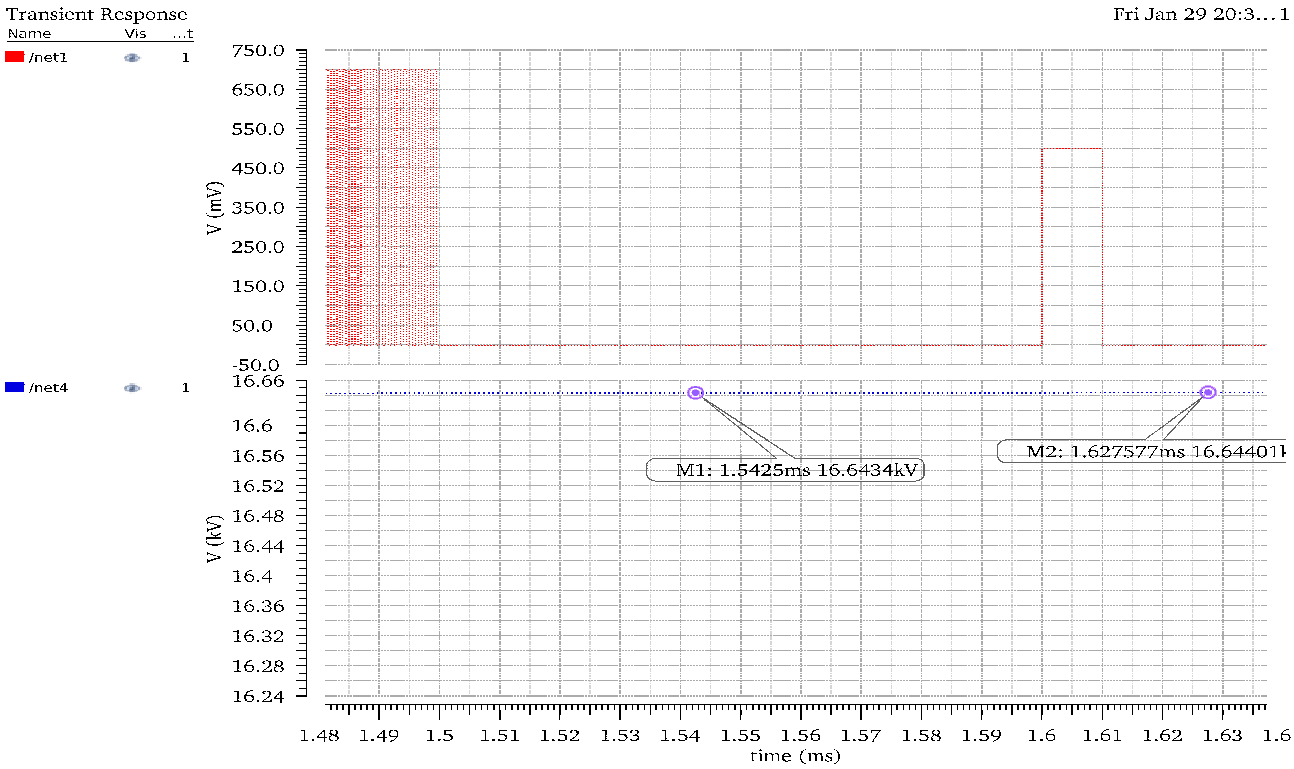


Figure 3.5: Memristance pertubation upon read

partially selected memristors on the bitlines were brought back to their initial RS however, the partially selected memristors on the wordline were perturbed and so were the other unselected memristors albeit by a small value.

This algorithm was conducted on a larger 7x7 crossbar and yielded the same results. To conclude this section, existing line biasing schemes cannot eliminate the effects of sneak currents in selectorless memristor crossbar arrays and as this wasn't the objective of the project it wasn't pursued any further.

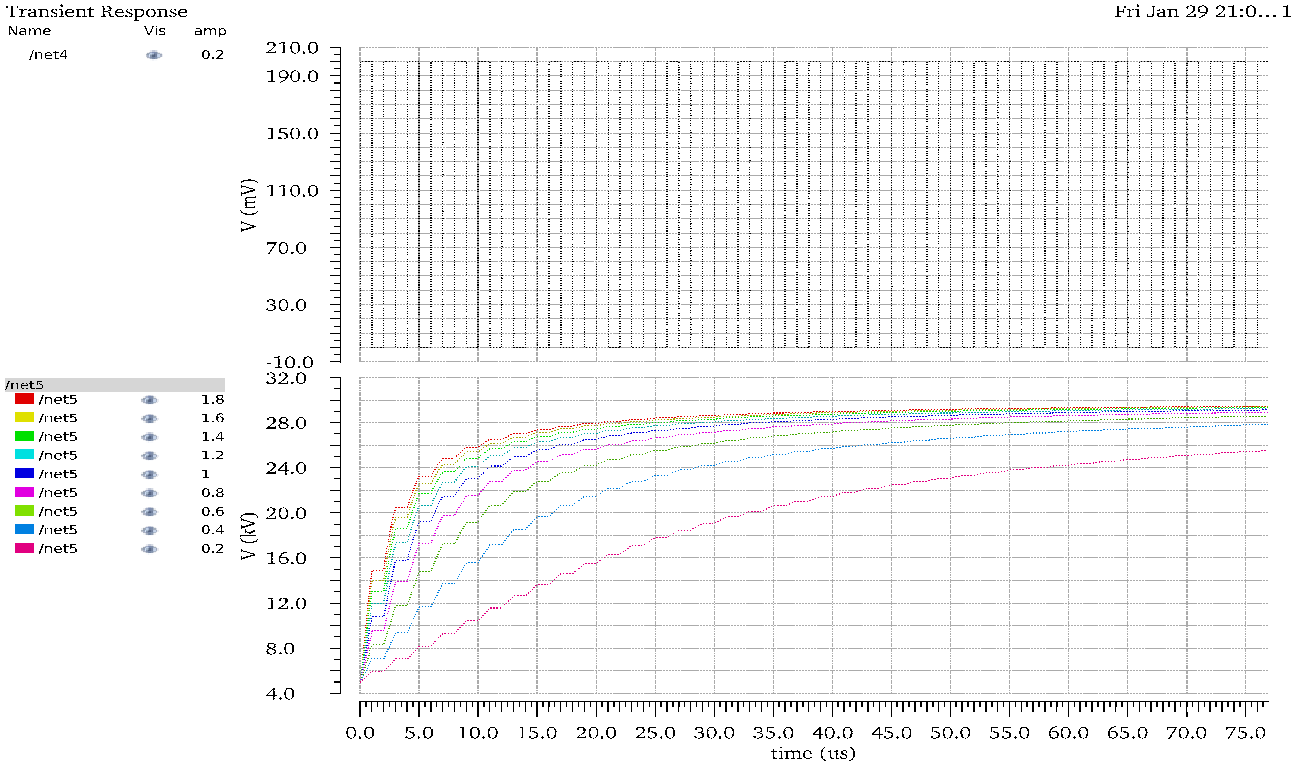


Figure 3.6: Memristance modulation from 5K-30K Ω using pulses of different amplitude.

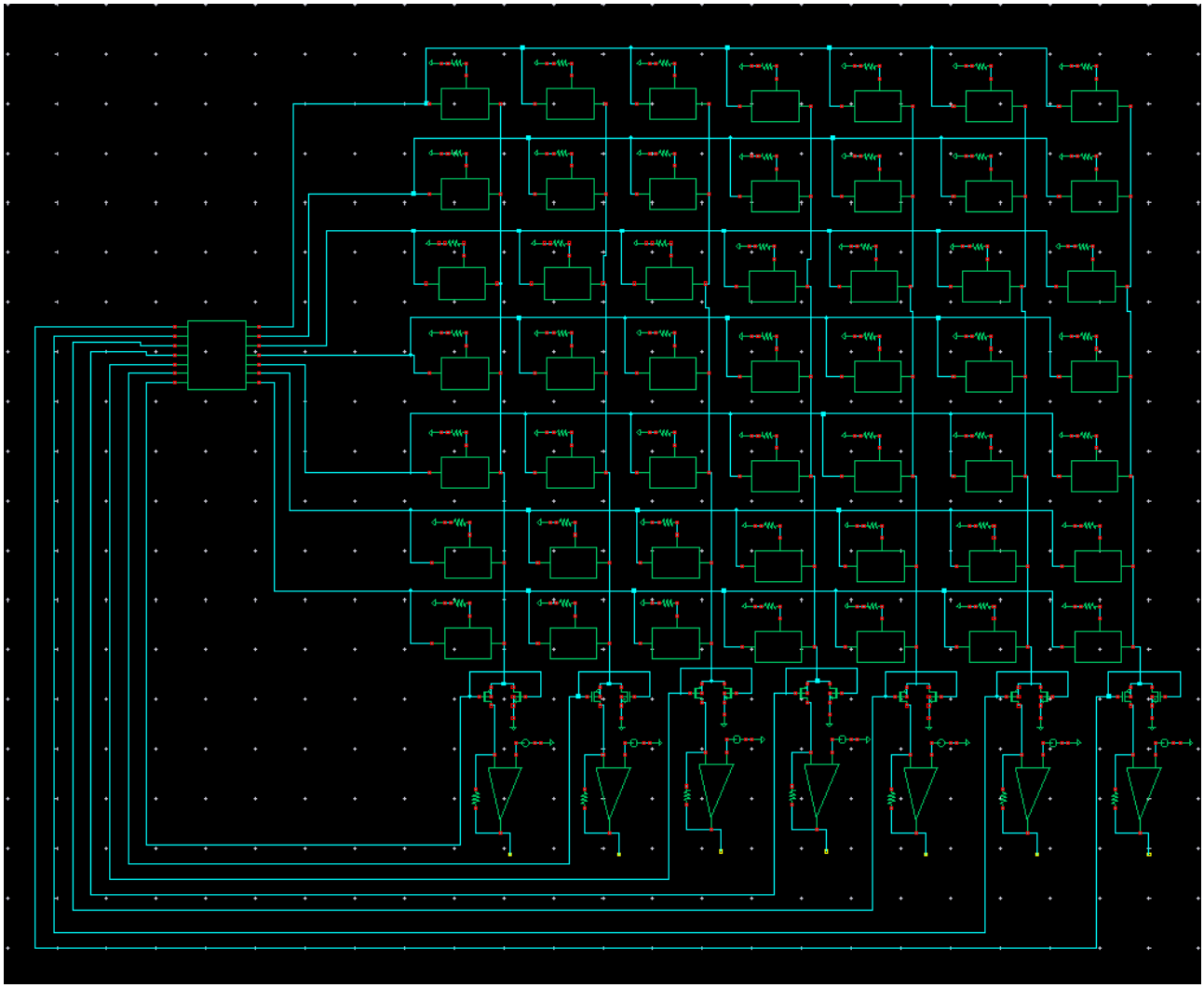


Figure 3.7: 7X7 memristor crossbar with a control module and a current sensing mechanism on each bitline.

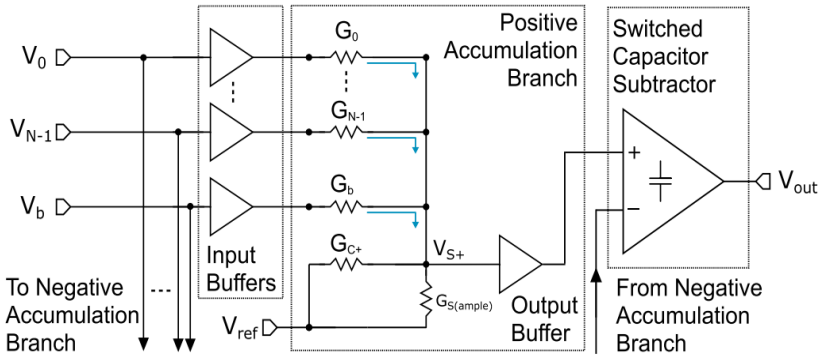


Figure 3.8: Switch capacitor based read circuit [36].

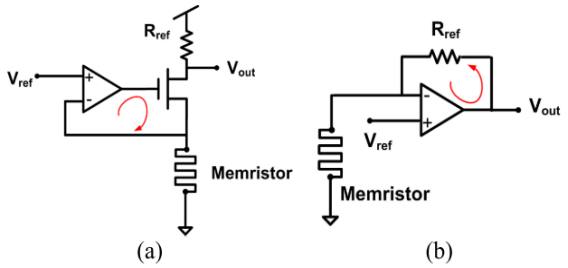


Figure 3.9: (a) Voltage divider based read circuit schematic. (b) Transimpedance amplifier based read circuit schematic, [37].

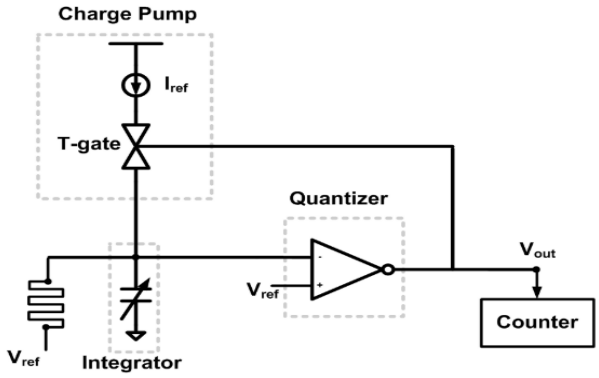


Figure 3.10: Sigma Delta read circuit schematic [37].

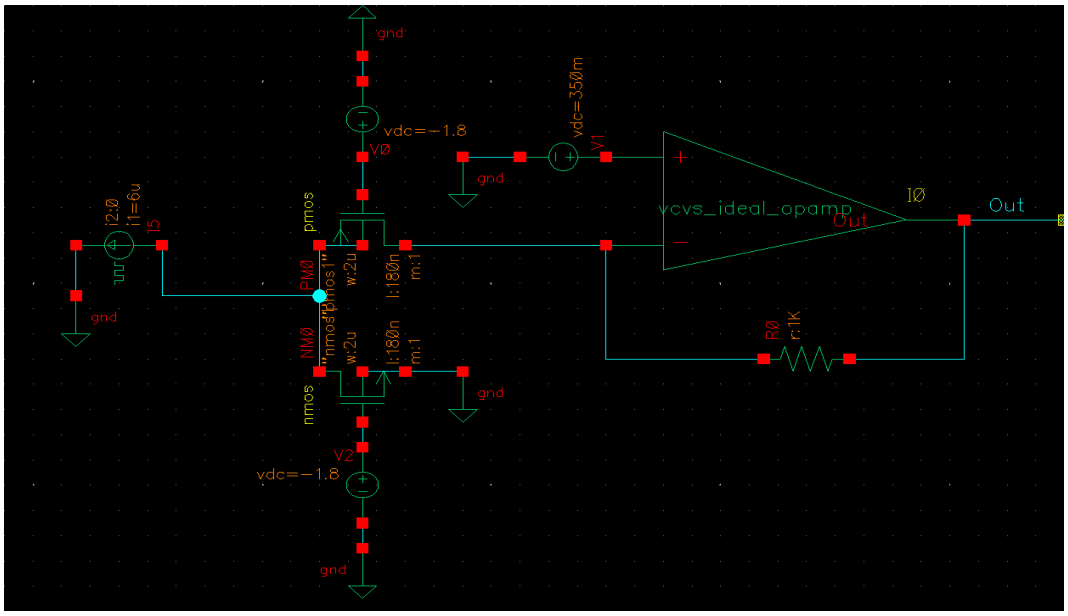


Figure 3.11: VCVS testbench.

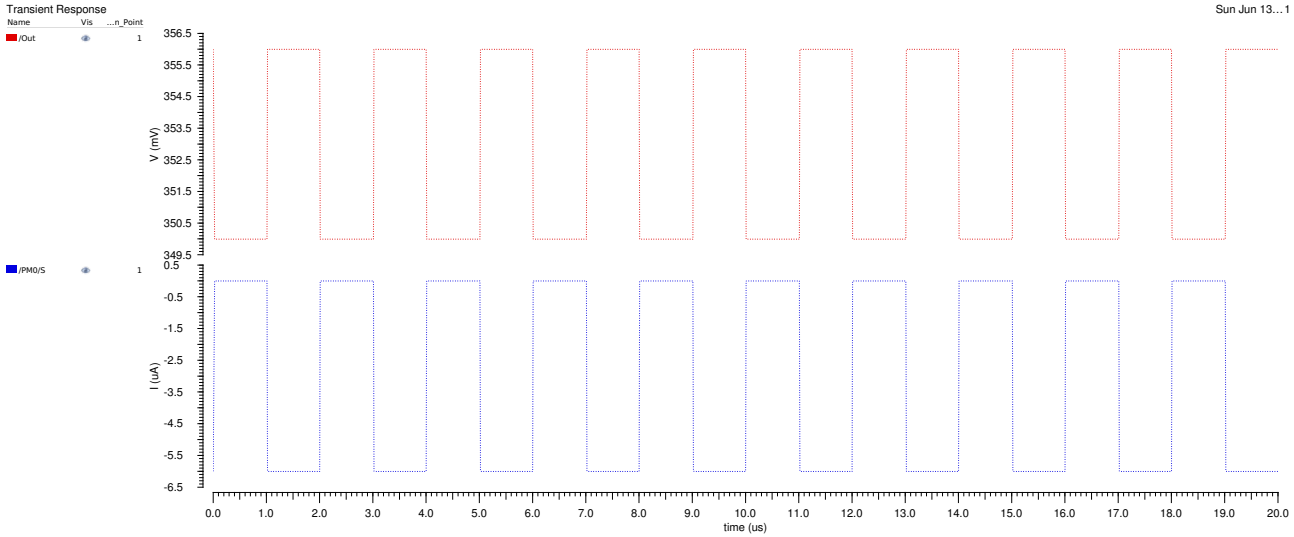


Figure 3.12: Graph showing the ideal behaviour of the TIA amplifier, top curve is the current flowing into the amplifier, bottom curve is the voltage output.

Figure 3.13: Graph demonstrating writing and reading to a memristor on a crossbar. The first plot demonstrates the RS of the memristor. The second plot shows the series of pulses applied to the target memristors wordline. The third plot shows the voltage pulses that allow the bitlines to be directed from ground to the TIA. And the fourth and fifth plots show the currents flowing to the N/PMOS respectively. The final plot shows the output of the TIA.

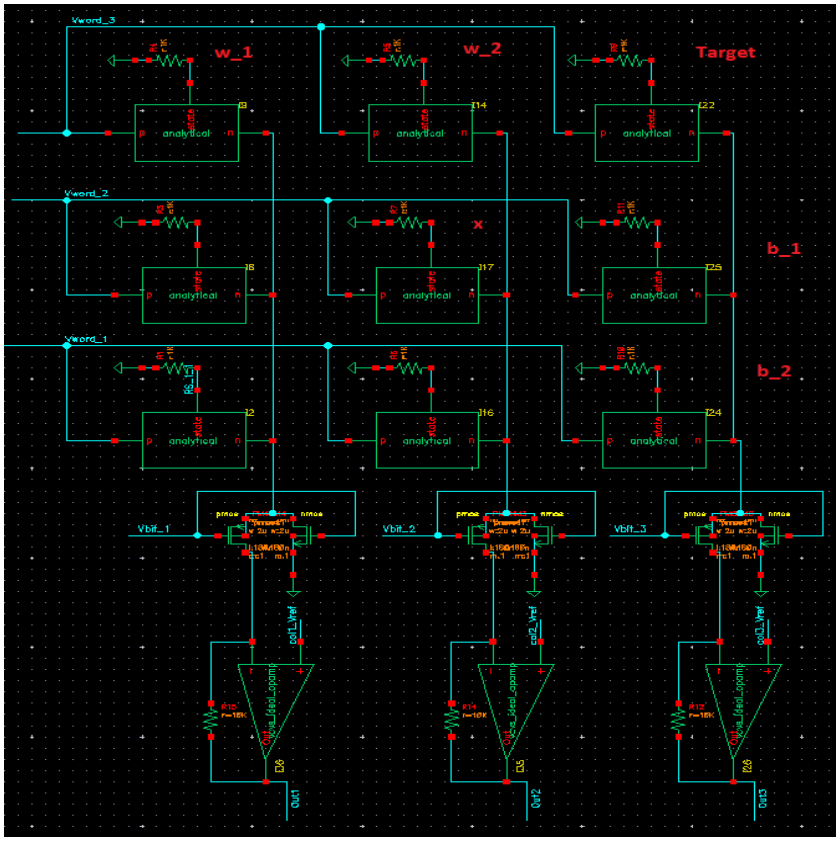


Figure 3.14: 3X3 selectorless memristor NN.

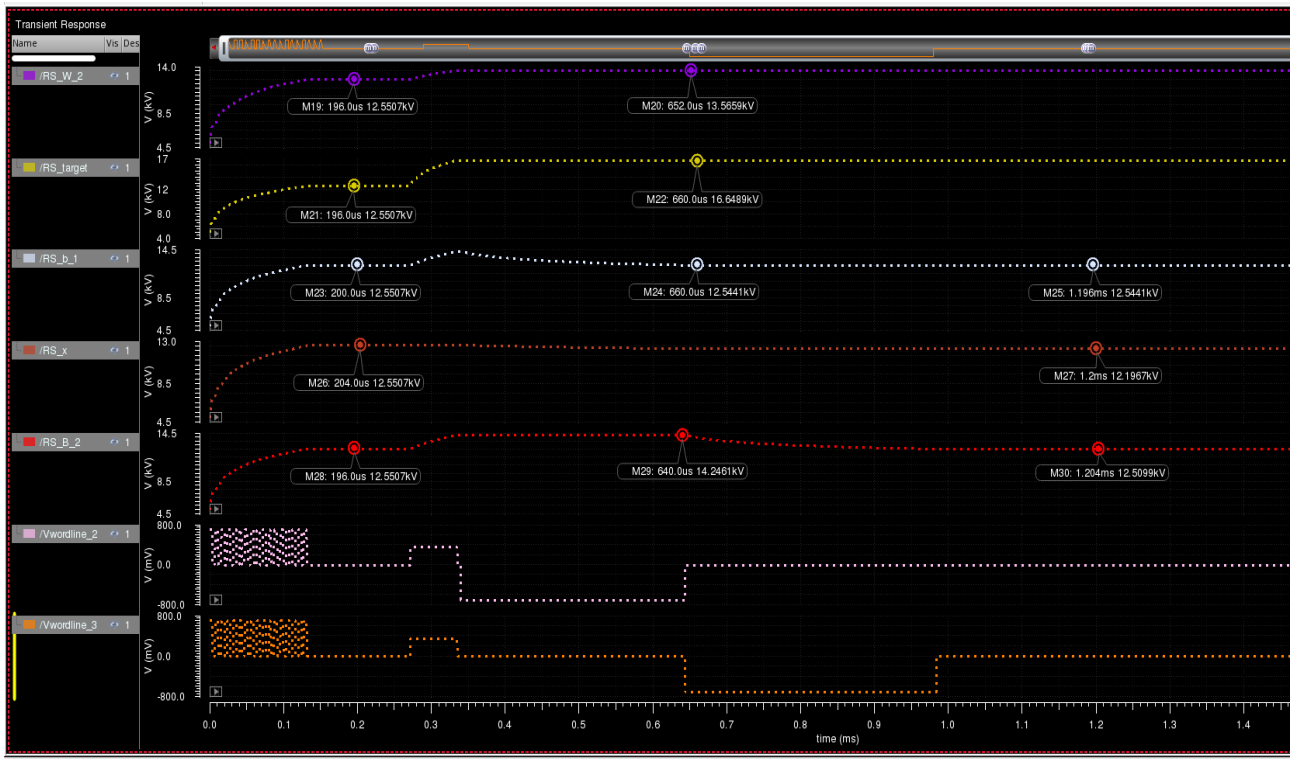

Transient Response
Name
Vis
Des
/RS_W_2
/RS_target
/RS_b_1
/RS_x
/RS_B_2
/Vwordline_2
/Vwordline_3
M19: 196.0us 12.5507kV
M20: 652.0us 13.5659kV
M21: 196.0us 12.5507kV
M22: 660.0us 16.6489kV
M23: 200.0us 12.5507kV
M24: 660.0us 12.5441kV
M25: 1.196ms 12.5441kV
M26: 204.0us 12.5507kV
M27: 1.2ms 12.1967kV
M28: 196.0us 12.5507kV
M29: 640.0us 14.2461kV
M30: 1.204ms 12.5099kV
V (kV)
V (mV)
time (ms)

Figure 3.15

Chapter 4

Requirements Capture

Besides exhibiting synaptic like properties, using memristors over traditional CMOS technology in neuromorphic engineering offers the following benefits:

- **Non-volatility:** Memristors have the ability to retain and remember their last state after power has been switched off for a time period of about 10 years.
- **Fast access:** Fast read and write access times of about 0.3ns [30] which minimizes power consumption.
- **High-Density:** Density measures the amount of binary information that can be stored on a given surface area. High density allows large amount of data to be stored in a small area. A capacity of up to $466GB/cm^2$ can be achieved with a memristor-based ReRAM using 5nm cells and $116GB/cm^2$ using 10nm cells, according to [31]. Memristor have a footprint of $4F^2$ (F is the minimum feature size) and can be further reduced to $4F^2/n$ by stacking n-layers of arrays or via $3D$ integration.

Significant efforts have been put in developing a memristor based neural network processor. In [32] a single layer pattern recognition system based on off-chip (sometimes called offline or ex-situ) training was designed. In this method, a neural network was trained using traditional software based methods and the corresponding memristive weights are programmed to the memristor cells in the crossbar array. Although successful, this approach relies on a traditional CPU which cannot cope up with the overwhelming increase in the amount of data to be processed nowadays. In addition, the memristor resistance values drift from the programmed values during normal read operation, thus requiring regular weight updating for proper operation. To overcome this, on-chip (sometimes called online or in-situ) training has been recently introduced [33] to allow constant updating of memristor weights and has led to improvements in computation time and power.

Nevertheless, these methods rely on selector devices (1T1R structures) to address a particular memristor when reading or writing data and thus limit device density and a fully neural architecture from being realised. To date, no machine learning device that uses a selectorless crossbar architecture to implement a neural network has been designed as the non-ideal effect of sneak currents on processing has deterred IC designers. Therefore, the goal of this project is to provide experimental evidence to support the claim that selectorless memristor neural networks can perform machine learning computations reliably and to investigate the networks performance and behaviour.

This will be done by using ideal circuit blocks to design an architecture that replicates a software based model to ensure any differences in performance can be attributed to the effect of sneak currents. The remainder of this section will introduce the dataset and the NN model that will be replicated to make it clear to the reader of what is expected of the memristor NN.

4.1 Dataset

Several datasets are used to benchmark the performance of machine learning models however these models require large networks to achieve a good accuracy. Thus, to keep the network size relatively small while achieving a good accuracy a simple labelled image dataset was created. The dataset consists of 1200 shuffled 4x8 binary pixel images of numbers in the range 0-5 generated by introducing 3 bit errors to their corresponding 'clean' images as shown in Figure 4.1. The code used to generate a single number is shown in Appendix B.

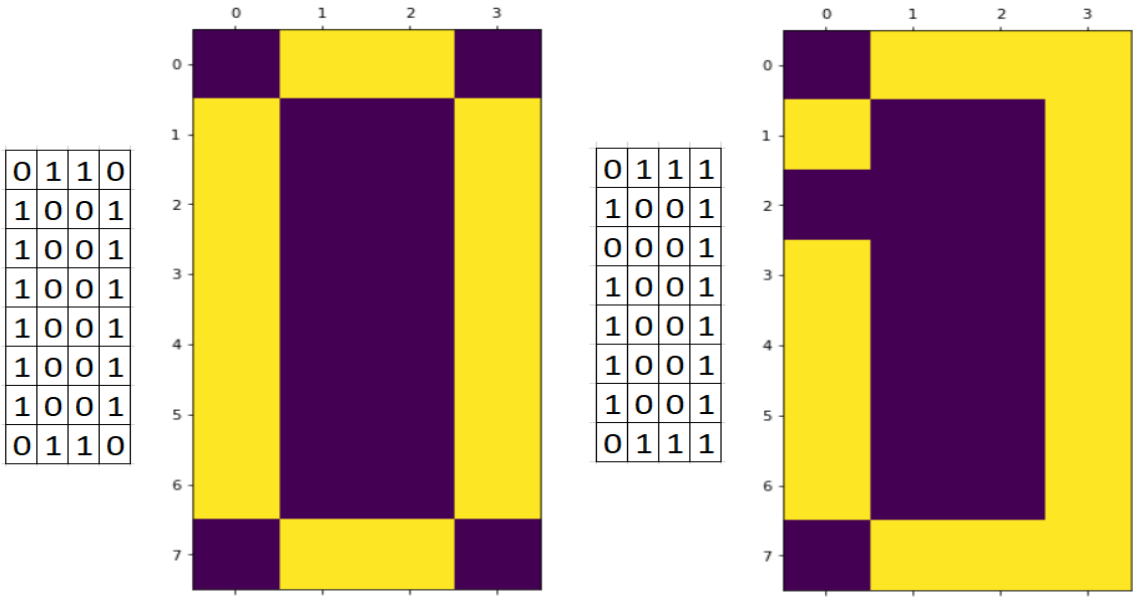


Figure 4.1: Example form image dataset. LHS shows a clean image, RHS shows the LHS image with 3 bit errors.

The dataset is then split into 900 training data and 300 test data and before being passed to the model, the image pixels are resized to take on binary values of either 0 or 0.5 rather than 0 or 1 for reasons described later.

4.1.1 Neural Network Model

A fully dense hidden layer NN model was selected and its structure can be visualised in Figure 4.2. Referring to the Keras code in Figure 4.3 that generated the model, the neurons in the hidden layer have a tanh activation function while the neurons in the output layer have a sigmoid activation function. In addition, the network uses 'Adam' as the optimizer to train the model and uses a categorical cross-entropy loss function. In addition, the network weights have been constrained between 0 and 1 and biases have been disabled to simplify the memristor crossbar. If negative weights were allowed, a separate bitline for negative weights and a counter per bitline

would have to be implemented in the memristor NN as in Figure 2.14.

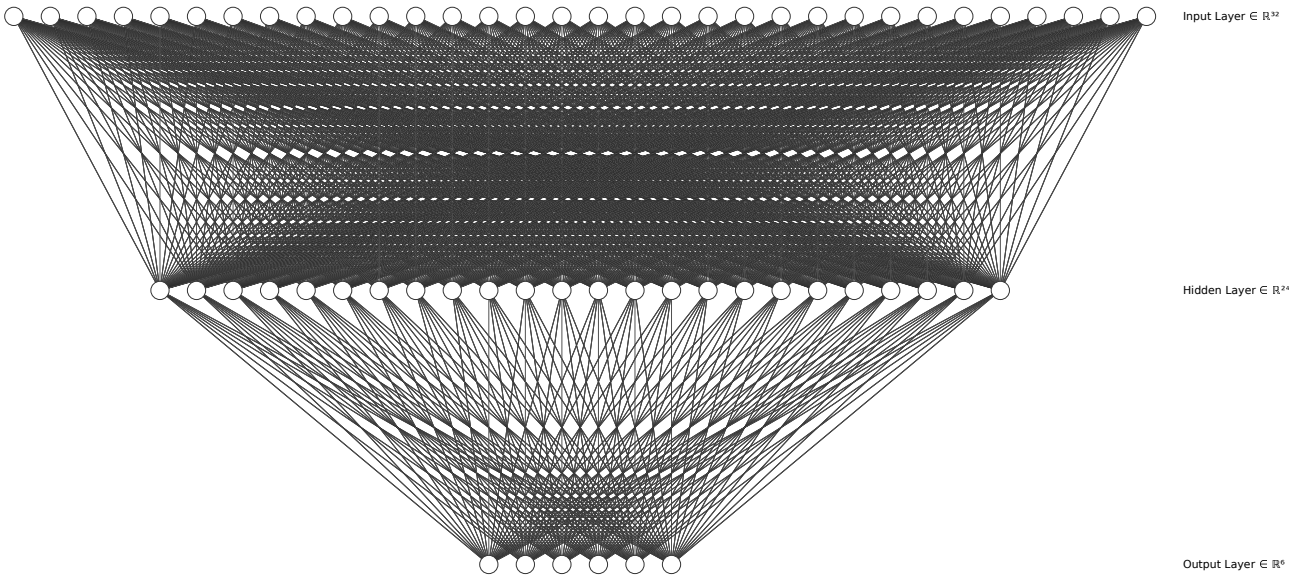


Figure 4.2: NN diagram with 32 pixels as inputs connected to a hidden layer consisting of 24 neurons that are in turn connected to an output layer consisting of 6 neurons. Each neuron in the output layer corresponds to a particular number, the largest output amongst the neurons determines the number detected.

```
model = Sequential()
model.add(Dense(24, activation='tanh', input_shape=(32,), kernel_constraint=Between(0, 1),
    use_bias=False))
model.add(Dense(6, activation='sigmoid', kernel_constraint=Between(0, 1), use_bias=False))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train_one_hot_encoded, batch_size=32, epochs=100)

```

Figure 4.3: Keras code used to generate NN.

The neural network achieves a classification accuracy of 97.5% and has the training profile shown in Figure 4.4.

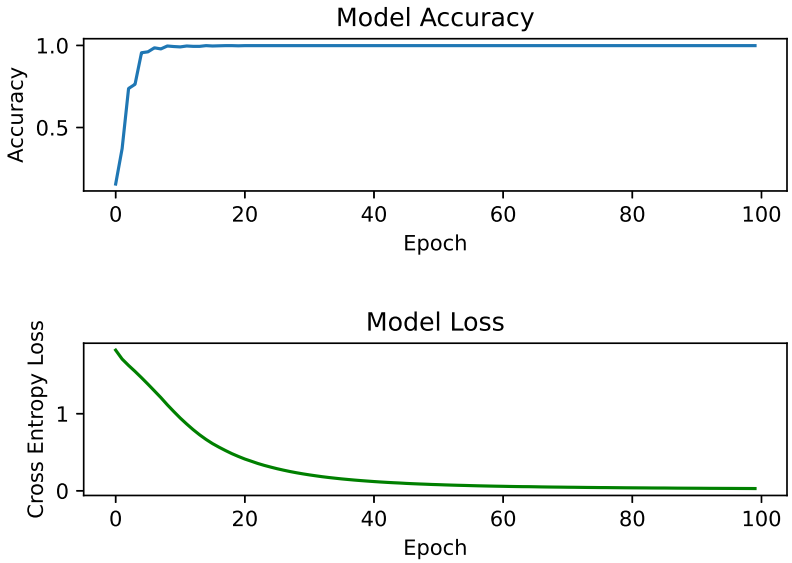


Figure 4.4: NN training profile

Chapter 5

Main Implementation

This chapter will explain the design and implementation of inference using selectorless memristor crossbar arrays that have an architecture based on the fully-connected NN presented in Chapter 4. The architecture presented tries to replicate the software based model as closely to try achieve the same classification accuracy.

The overall architecture can be visualised in Figure 5.1. The architecture consists of 2 crossbar arrays. The first crossbar array represents the hidden layer consists of 32 wordlines and 24 bitlines, comprising of 768 memristors and 24 neurons at the base of the bitlines. A Voltage corresponding to each of the 32 pixels from an image are passed to the crossbar's respective wordlines. The second crossbar array represents the final output layer consists of 24 wordlines and 6 bitlines, comprising of 144 memristors and 6 output neurons at the base of the bitlines. The outputs from the 24 hidden layer neurons are passed as inputs to the wordlines in the second crossbar array while the six neurons in the second crossbar array represents a particular number from 0 to 5.

Each neuron as shown in Figure 5.2, implements a MAC function followed by an activation function $I = \Theta\{\Sigma(V_i \cdot G_{ij})\}$ as explained in Section 2.8. A TIA implements the MAC function while the activation function Θ is implemented by a Verilog-A model. As in the software model, the neurons in the hidden layer implement a tanh function while the neurons in the final layer implement a sigmoid function. The Verilog-A code used to create the activation function modules is shown in Appendix C and the output from a basic testbench that runs a DC sweep through the module is shown in Figure 5.3.

Besides the aforementioned, a Verilog-A control module handles performing inference using the crossbar arrays and this module will be the focus of Section 5.2.

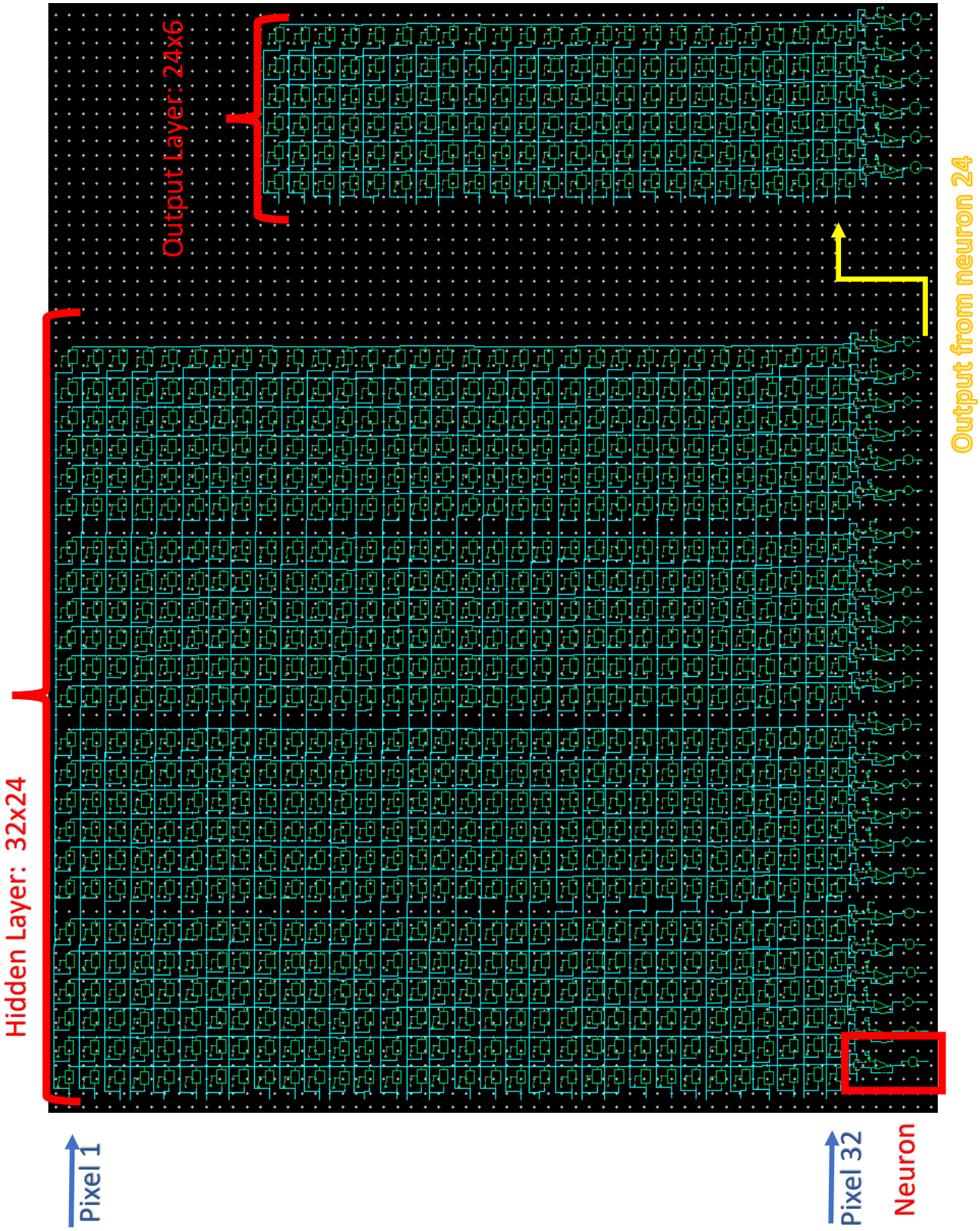


Figure 5.1: Memristor crossbars

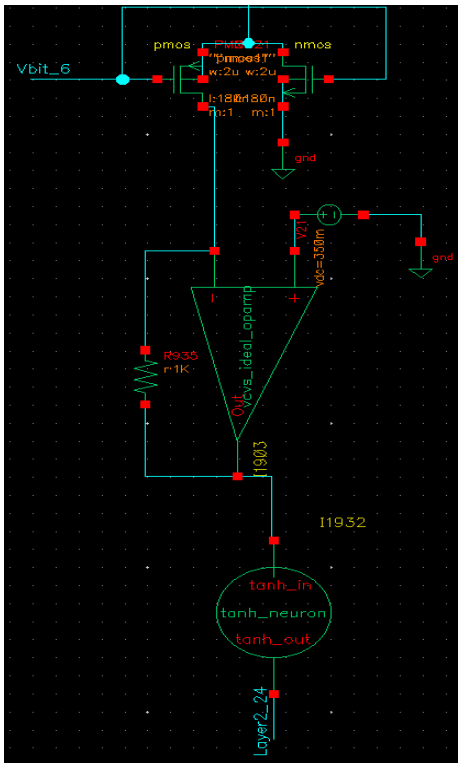


Figure 5.2: Neuron from the hidden layer consisting of a TIA connected to a tanh module.

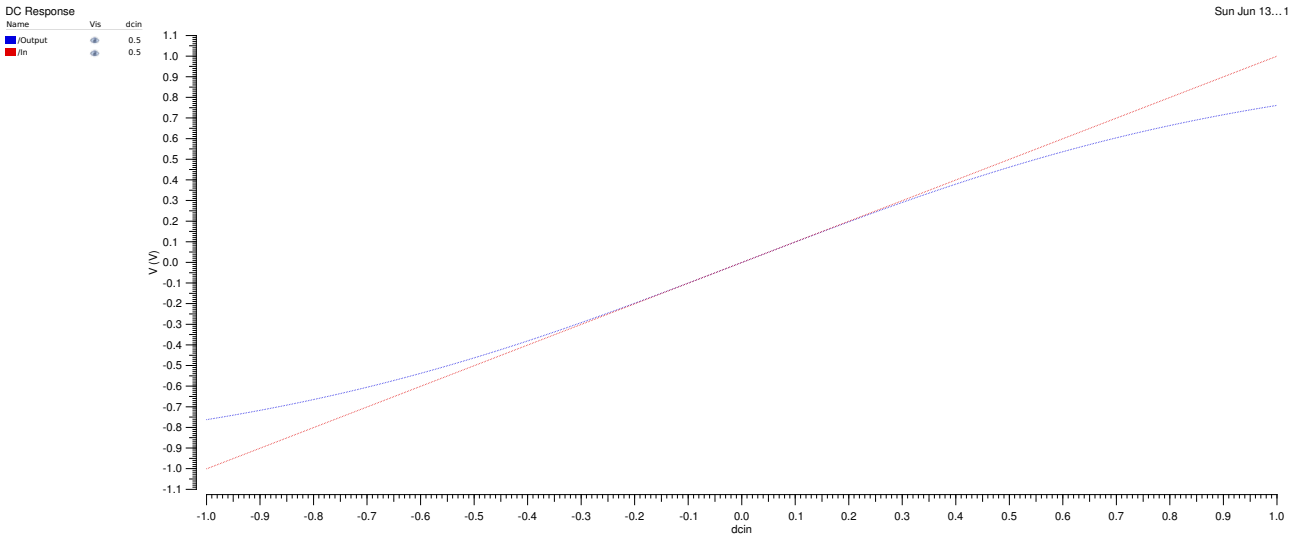


Figure 5.3: Graph showing a DC sweep through the tanh module from the range -1V to 1V.

5.1 Initializing the state of the memristors

As the objective of this project is to assess the computational capabilities of a selectorless memristor crossbar array, it was assumed that each memristor can be programmed to a RS that corresponds exactly to a weight by setting the initial RS of the memristor equal to the weight.

A detailed explanation of this process is beyond the scope of this report. To summarise, the method involves the following: The neural network is trained and the weights of the network are downloaded. As the weights exist within a range from 0 to 1 they are scaled to make full of the memristive RS range from 5k-30kΩ. Since, the analogous implementation of a MAC function $Y = \Sigma(W^T X)\}$ is $I = \Sigma(V_i \cdot \frac{1}{R_{ij}})$ the weights are scaled accordingly: $R = 30000 - (5000W)$ so that a memristor with a corresponding weight of 0 has an RS of 30000Ω while a memristor with a weight of 1 has an RS of 5000Ω. Each memristor has an initial RS variable that can be set to a parameter in Cadence. A special parameter extraction script is then run in Cadence to read from a text file a weight and set it as the initial RS of a respective memristor.

5.2 Control Module

The control module consists of various sets of pins as shown in Figure 5.4. The 'voltageOut' pins on the right hand side are connected to the wordlines of the first crossbar via the 'Vword' wires and each pin outputs a voltage that corresponds to the binary state of a pixel as described in Section 4. Moreover, the 'BitLineSet' pin on the left hand side is connected via the 'Vbit' wire to the N/PMOS of all the TIA in the various neurons. Since, inference is taking place the pin always outputs -1.8V so that current in the bitlines is always directed to the TIA rather than ground. Furthermore, the 'out' pins on the bottom are input pins that are connected to the six neurons in the second crossbar via the respective 'Final' wires. Lastly, the 'Clk', 'Load' and 'Sample' pins are not connected to the crossbar array but are rather used to test various signals within the control block that determine its operation.

The control module can be broken down into 3 different stages: Initialization, Crossbar Writing and Results Verification. Referring to the Verilog-A code listed in Figure 5.5, the initialization stage involves setting the clock frequency at which the system operates, followed by initializing the clock, load and sample signals using the timer function. The Timer function sets a future event to occur at a specified time either just once or repeating at a specified period. Thus, at every 2*Clock Frequency the clkstate variable toggles to create the clock signal while at every 4*Clock Frequency the 'loadstate' and 'samplestate' variables toggle to create the load and sample signals that have a frequency 2*Clock Signal. The transition statements convert these variables to electrical signals and further impose a condition on the signals such that the sample signal goes high when the clock signal is high and the load signal goes high when the clock signal is low. All three signals can be visualised in the graph shown in Figure 5.6 for the case where the clock frequency is 50MHz. Lastly, the memristor bitlines are directed to the TIA by turning on the PMOS.

The second stage involves writing to the crossbar. At every negative edge of the load signal a line from a text file containing an images 32 binary pixel values and the image's correct classification number is read and the data is stored in various variables. Four example lines from a text file that contains 300 test data values are listed in Figure 5.7. The first float from

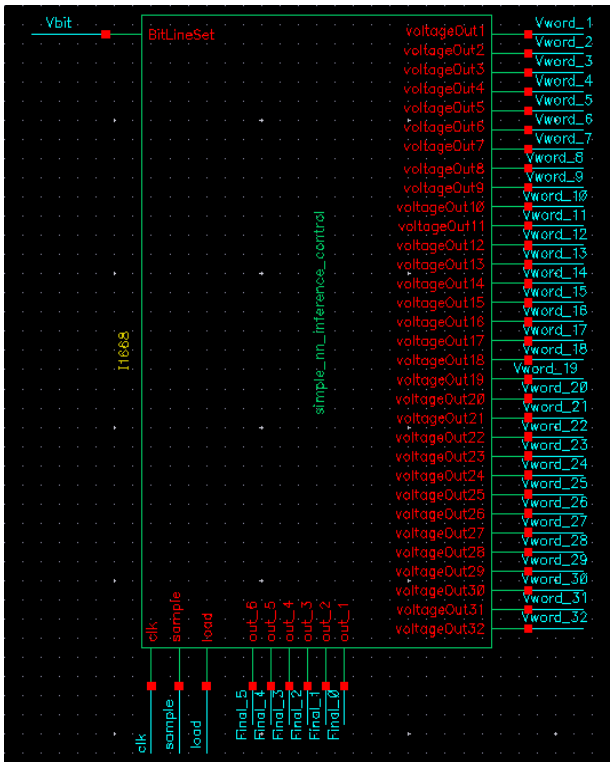


Figure 5.4: Symbol of the control module

each line represents the classified image number (in this case 0,5,2,1) while the remaining the 32 numbers represent the binary pixel values. Each pixels' value is then written to a wordline corresponding to the pixel number when the clock signal is high. Figure 5.8 shows the data from the text file lines being output as voltages to the first 5 wordlines of the crossbar. Note, when the clock is low the voltage on the wordlines is set to 350mV to prevent sneak currents in the crossbar as the TIA also biases the bitlines to 350mV.

The final stage involves verifying the results of the inference which takes place when the clock signal is high. At the falling edge of the sample signal, the six 'out' pins which are connected to the outputs of the neurons in the second crossbar (output layer) are sampled to determine which neuron has the highest output voltage. If the neuron that outputs the highest voltage corresponds to the actual image classification number (first float on each line in the text file) a 'correct' detection is registered or else a 'false detection' is registered. The ratio of the total number of correct classifications to total test samples determines the inference accuracy. Figure 5.9 demonstrates the output from the six neurons on the text file example shown in Figure 5.7 while Figure 5.10 shows the output from the Cadence log file which tracks the inference at every clock cycle.

```
// Initialization
parameter real Fclk = 50M; //clk frequency
real Wclk; //pulse width
real Wsample;
real true_result,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,v,W,X,Y,Z,A1,B1,C1,D1,E1,F1;

integer clkstate;
integer samplestate;
integer loadstate;
integer counter;
integer result;
integer fidX,
integer correct;

analog
begin
@(initial_step)
begin
  Wclk = (1/Fclk)/2;
  Wsample = (1/Fclk)/4;
  clkstate = 1;
  samplestate = -1;
  counter = 0;
  correct = 0;
  fidX = $fopen("full_data.txt", "r");
end

@(timer(0, Wclk))
begin
  clkstate = -clkstate;
  counter = counter + 0.5;
 end
@(timer(0, Wsample))
begin
  samplestate = -samplestate;
  loadstate = -loadstate
end

V(clk) <+ transition( (clkstate>0)?1:-1, 0.01n, 0.01n, 0.01n);
V(sample) <+ transition( (clkstate>0 && samplestate>0)?1:-1, 0.01n, 0.01n, 0.01n);
V(load) <+ transition( (clkstate<0 && loadstate>0)?1:-1, 0.01n, 0.01n, 0.01n);

//Memristor bitlines directed to transimpedance amplifier
V(BitLineSet) <+ -1.8;

```

```
\\Crossbar Writing
@(cross(V(load),-1,0.01n))
begin
  $fscanf(fidX, "%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
    %f %f %f %f %f %f %f %f",true_result,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,v,W,X,Y,Z
    ,A1,B1,C1,D1,E1,F1);
end

V(voltageOut1) <+ transition( (clkstate>0)?A:0.35, 0.01n, 0.01n, 0.01n);
V(voltageOut2) <+ transition( (clkstate>0)?B:0.35, 0.01n, 0.01n, 0.01n);
V(voltageOut3) <+ transition( (clkstate>0)?C:0.35, 0.01n, 0.01n, 0.01n);
|
| "Repeat for all 32 wordlines"
|
V(voltageOut32) <+ transition( (clkstate>0)?F1:0.35, 0.01n, 0.01n, 0.01n);

\\ Results Verification
if(clkstate>0 && V(out_1) > V(out_2) && V(out_1) > V(out_3) && V(out_1) > V(out_4) && V(
    out_1) > V(out_5) && V(out_1) > V(out_6))
  result = 1;
else if(clkstate>0 && V(out_2) > V(out_1) && V(out_2) > V(out_3) && V(out_2) > V(out_4) && V
    (out_2) > V(out_5) && V(out_2) > V(out_6))
  result = 2;
else if(clkstate>0 && V(out_3) > V(out_1) && V(out_3) > V(out_2) && V(out_3) > V(out_4) && V
    (out_3) > V(out_5) && V(out_3) > V(out_6))
  result = 3;
else if(clkstate>0 && V(out_4) > V(out_1) && V(out_4) > V(out_2) && V(out_4) > V(out_3) && V
    (out_4) > V(out_5) && V(out_4) > V(out_6))
  result = 4;
else if(clkstate>0 && V(out_5) > V(out_1) && V(out_5) > V(out_2) && V(out_5) > V(out_3) && V
    (out_5) > V(out_4) && V(out_5) > V(out_6))
  result = 5;
else if(clkstate>0 && V(out_6) > V(out_1) && V(out_6) > V(out_2) && V(out_6) > V(out_3) && V
    (out_6) > V(out_4) && V(out_6) > V(out_5))
  result = 6;
else result = 0;

@(cross(V(sample),-1,0.01n))
  begin
    $display("counter=%d", counter);
    $display("            True result=%f", true_result);
    if((result-1) == true_result)
    begin
      correct = correct +1;
      $display("            Number of Correct=%d", correct);
    end
    else ($display("   FALSE DETECTION !!!"));
    $display("            Number detected=%d", result-1);
  end

@(final_step)
begin
  $fclose(fidX);
end

```

Figure 5.5: Verilog-A code implementing the econtrol module

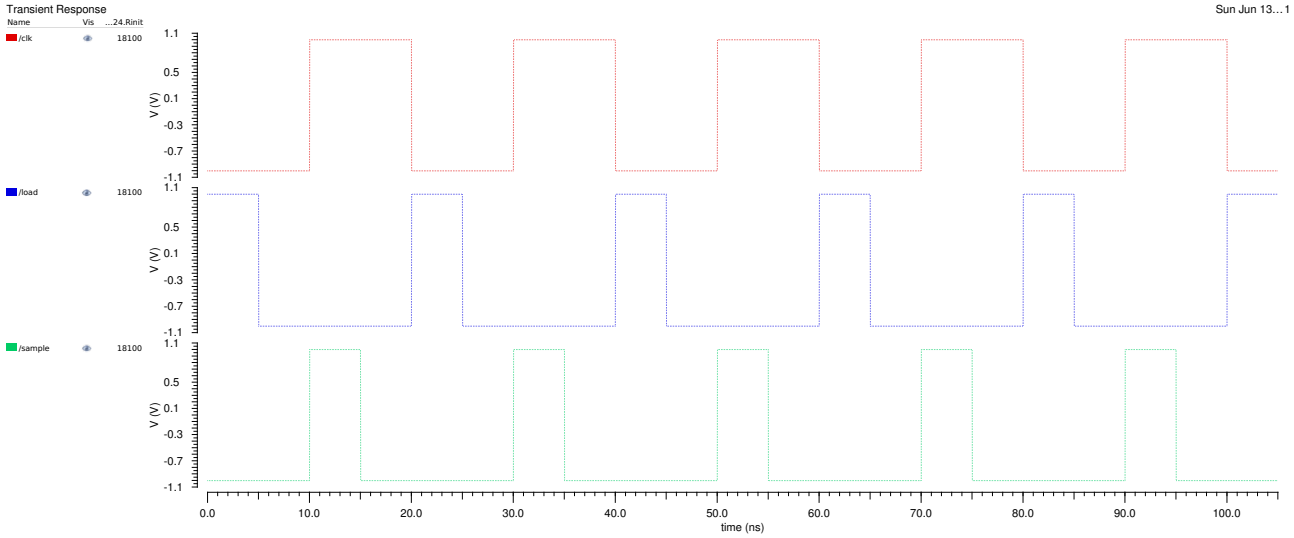


Figure 5.6: Graph showing clock, load and save signals.

```
0.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0
    0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0
5.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0
    0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0
2.0 1.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0
    0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
    1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0

```

Figure 5.7: Four example data samples from the est data set.

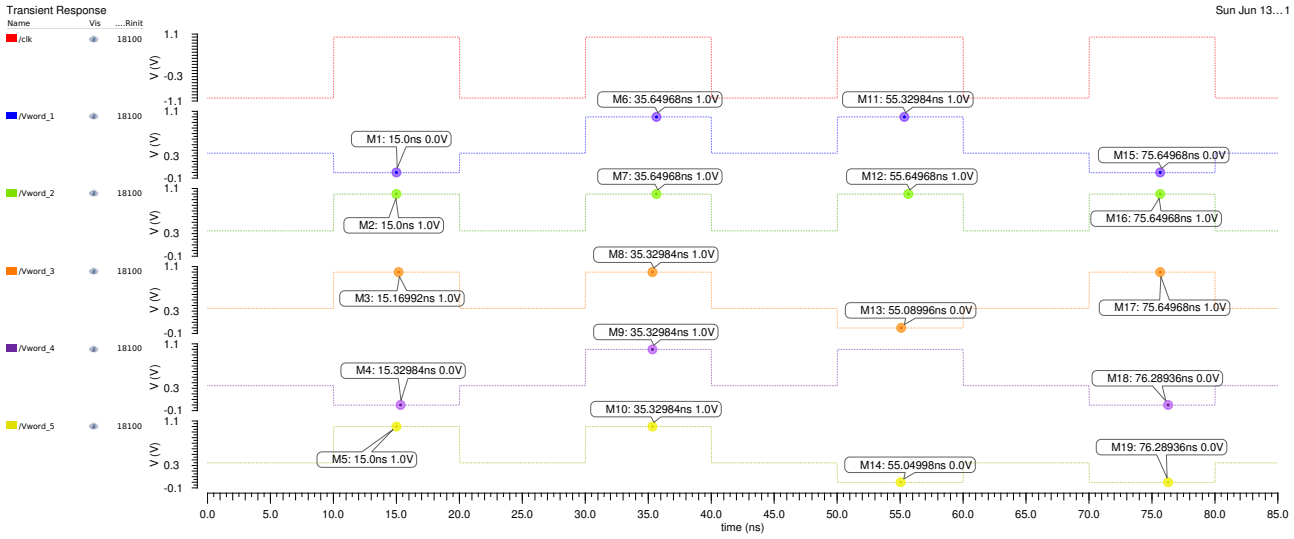


Figure 5.8: Graph showing the wordline voltages corresponding to the data from the text file when the clock signal is high.

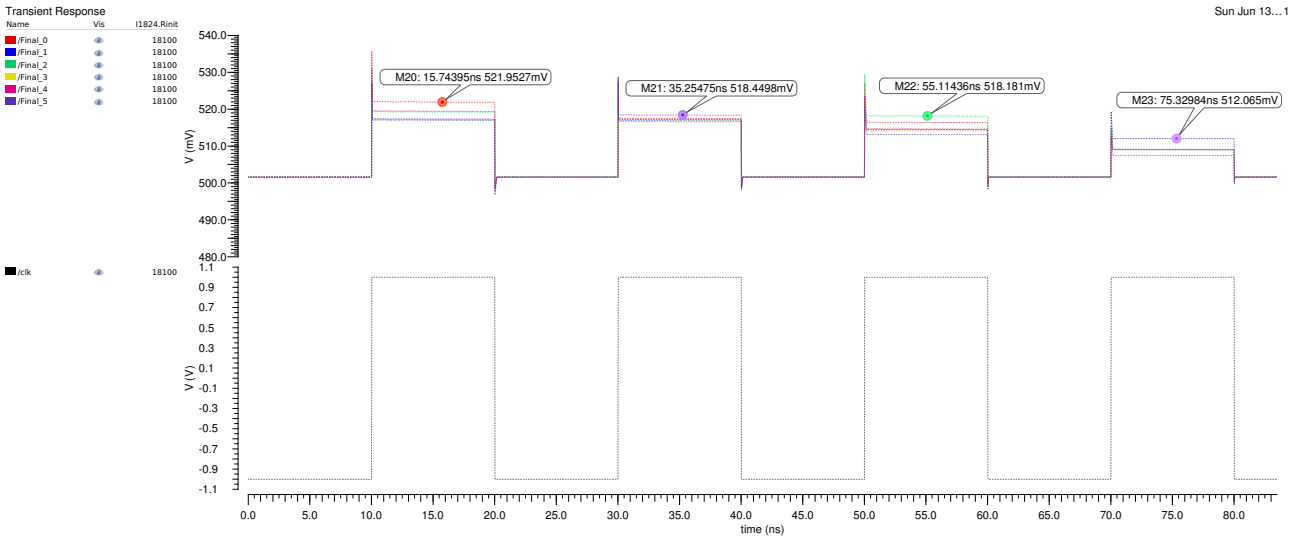


Figure 5.9: Graph showing voltage output from the six output neurons. The markers show the highest voltage output from a particular neuron.

```
counter=2
                                                    True result=0.000000
         FALSE DETECTION !!!
                                                    Number detected=4
counter=4
                                                    True result=5.000000
                                                    Number of Correct=1
                                                    Number detected=5
counter=6
                                                    True result=2.000000
                                                    Number of Correct=2
                                                    Number detected=2
    tran: time = 75 ns        (7.5 %), step = 1.631 ns     (163 m%)
counter=8
                                                    True result=1.000000
                                                    Number of Correct=3
                                                    Number detected=1
```

Figure 5.10: Output log from Cadence terminal.

Chapter 6

Testing and Results

This chapter details the various tests performed on the selectorless crossbar memristive NN under different system parameter constraints and analyses its behaviour by comparing its performance to the software model.

Note: both the memristor NN and software model were tested on a dataset of 300 samples and when 'Classification Accuracy' is reported it refers to the 'total number of correctly classified data samples/300'.

6.1 Network Inputs

This section investigates the performance of different binary input values to the network classification accuracy. The images can take on any binary values, for example: 0 or 1, 0 or 0.5 and these binary values are converted to voltages on the wordlines. Figure 6.1 shows a plot of the relationship between pixel values and network classification accuracy. To obtain the classification accuracy for a particular binary pixel value, the software model was trained with a dataset that had the same binary values and the weights were downloaded and used to set the corresponding memristors to an initial RS.

The software classification accuracy remains relatively constant and the slight variation among the values could be attributed to the software model weights diverging around the global minimum since an adaptive learning rate (see Figure 6.2) wasn't set when training the software model. The memristor NN classification accuracy also reflects this upto 0.9V beyond which the network performance drastically decreases as the pixel value increases. This is because the change in a memristors RS according to Eqn 3.2 is exponentially dependent on the magnitude of the voltage applied on it. Thus, large voltages applied on the wordlines will perturb the state of the memristors and will also lead to larger sneak currents flowing that will also cause the memristors state to be perturb. Note: As in later sections the RS of a memristor cannot be tracked to validate this change since the network weights differ depending on the binary pixel data set used to train the the software model..

Notice the binary pixel range, it was limited to 0.4V. This is because the memristor bitlines are held at 0.35V and thus the wordlines need to be held at a voltage slightly higher for inference to take place. Based on the aforementioned, a meaningful range over which inference can take place to a high accuracy can be established - 0.4V to 0.9V. Therefore in the future,

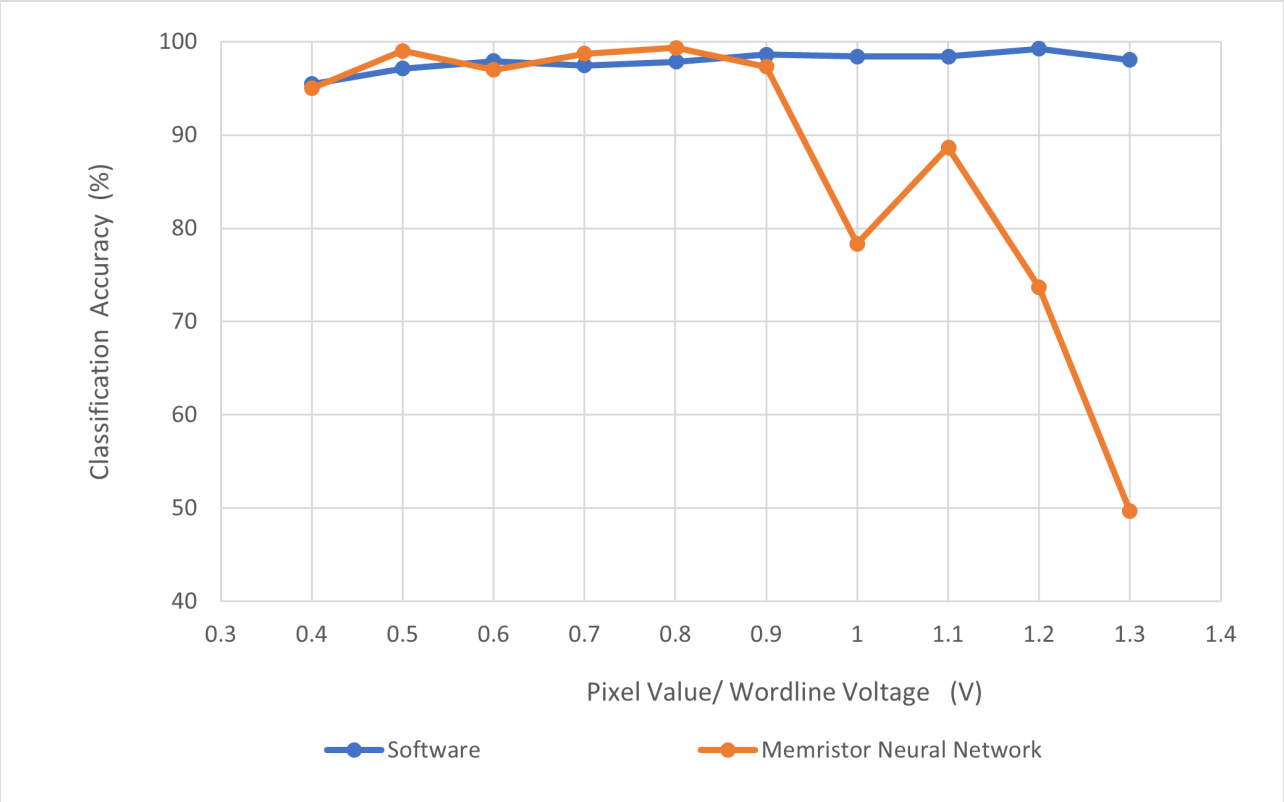


Figure 6.1: Graph showing software and memristor neural network classification accuracy vs binary pixel value.

when a selectorless crossbar is part of an IC where a user can program it to perform any function, an IC designer should constraint the inputs to the wordlines by placing a block at each line that implements the following function in Eqn 6.1. To verify this, the memristor NN was programmed with weights that correspond to pixel values of 1.3V but was tested with a dataset where the pixel values were limited to 0.9V. The network classification accuracy improved from 49.67% to 99.33%.

$$y = \begin{cases} 0.4 & x < 0.4 \\ x & 0.4 < x < 0.9 \\ 0.9 & x > 0.9 \end{cases} \tag{6.1}$$

In the software model, when a pixel has an input of 0, all the neurons that it is connected to receive no contribution since W*0 = 0. The analogous implementation to this in a crossbar would be to set the wordline to the same volatage as the bitline - 0.35V. Thus, the pixel values/data passed to the wordlines should be '0.35 or 0.35+X' rather than '0 or X' (where X is the on binary value), as seen in Figure 6.4 and Figure 6.3. This limits the current contribution from a particular wordline that has a pixel input of 0 and hence limits sneak currents from flowing. This was the initial method that was implemented however it resulted to lower classification accuracies as seen in Table 6.1. To ensure a fair comparison, the memristors in the NN were initialised to the same weights and the same test dataset was used.

From the output log, I noted that the network seemed to always fail to distinguish between the numbers 3 and 4 and 5 and 4 which leads me to postulate that perhaps the sneak currents that

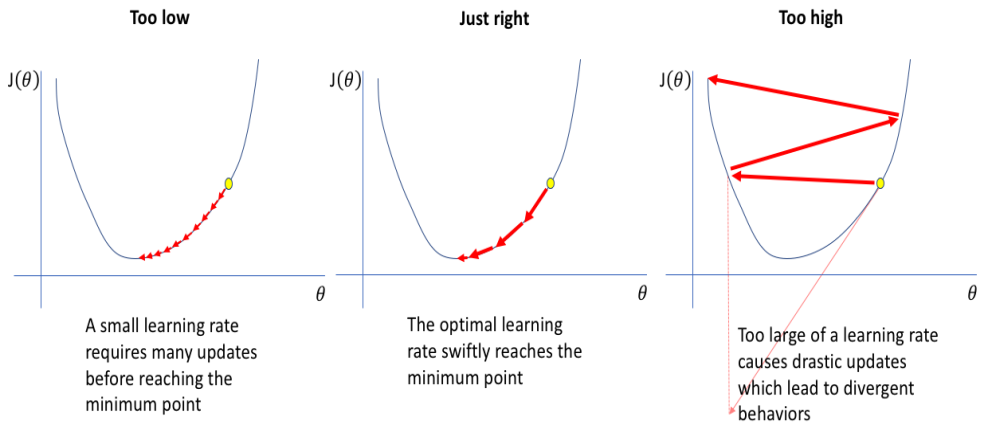


Figure 6.2: Various diagrams showing the effect of various learning rates on converging to the optimum network weights [40].

flow when the input is 0 rather than 0.35 improves the inference accuracy of the network. In addition, from Figure 6.1 the memristor NN has a higher classification accuracy in comparison to the software model even though both NN have the same weights. One reason that could possibly explain this is stochastic resonance [41], a phenomenon that involves adding white noise to a signal that is normally too weak to be detected by a sensor. The frequencies in the white noise corresponding to the original signal's frequencies will resonate with each other, amplifying the original signal while not amplifying the rest of the white noise – thereby increasing the signal-to-noise ratio, which makes the original signal more prominent. The binary pixel input is this case is the signal/pattern that needs to be detected and when passed to the crossbar directs the 'noise' i.e sneak currents towards a particular bitline/neuron depending on the image it represents.

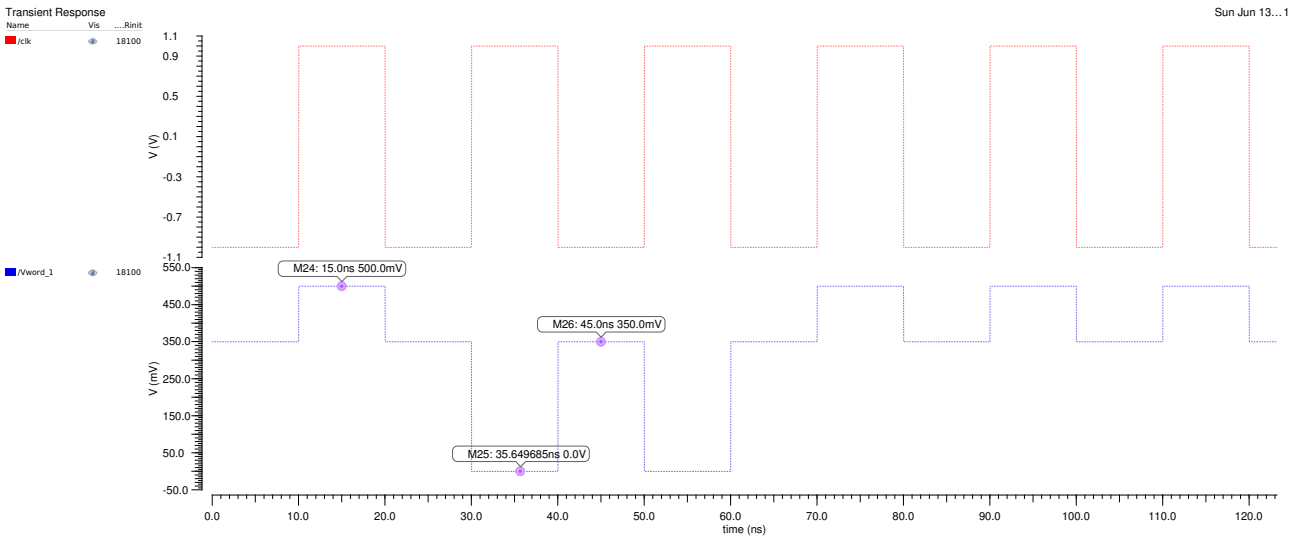


Figure 6.3: Plot showing the voltage applied on the first wordline where each pulse corresponds to the first pixel and 6 data samples are shown. The 0 or X scheme has been implemented where X is 0.5.

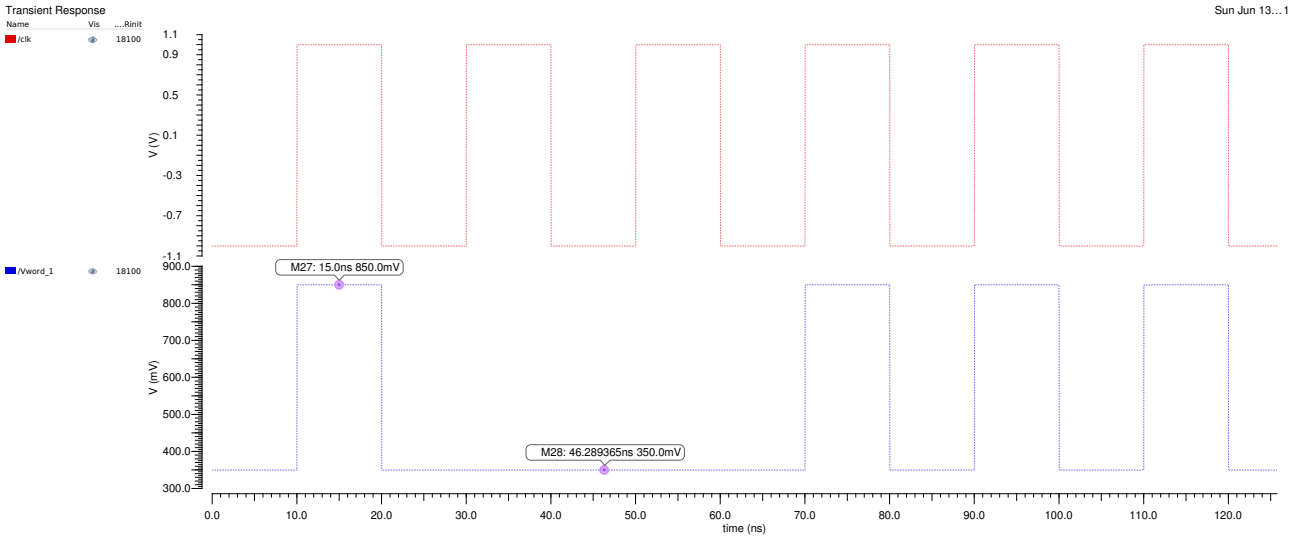


Figure 6.4: Plot showing the voltage applied on the first wordline where each pulse corresponds to the first pixel and 6 data samples are shown. The '0.35 or 0.35+X' scheme has been implemented where X is 0.5.

Table 6.1: Comparing classification accuracies with or without 0.35V bias.

	Classification Accuracy (%)	
Binary Pixel Value (X)	0 or X	0.35 or 0.35+X
0.4	78.7	95
0.5	77.3	99

6.2 Network Weights

Currently, fabricated working memristors have a RS range of 5K-30KΩ and thus in the previous section the memristors in the various tests were set to a RS within this range. However, as more efforts are put into developing memristors, in the future, memristors that can have a wider RS ranges could exist. This section begins by investigating the inference performance of the network when the memristors are constrained to different RS ranges.

Table 6.2 and Table 6.3 shows the NN classification accuracy under different binary pixel inputs when the memristors are constrained to different RS ranges. To ensure a fair comparison of results, the software based model was trained once and the weights (within a range of 0 to 1) were saved and then scaled accordingly to take on the various RS ranges. Moreover, each test was conducted with the same test data set. Thus, the only difference among the tests was the scaling function applied to the original weights. From the results, it can be seen that highest classification accuracy for the different binary inputs occurs at different RS ranges. This implies that the difference between the orange and blue plot in Figure 6.1 can be bridged for inputs larger than 0.9V by finding the optimum RS-range corresponding to an input.

When fabricating large arrays of memristor crossbars a few memristors will not be functioning due to process variations and are said to be 'dead'. That is, no current can flow through them (oxide filaments cannot form) creating open circuits in the crossbar array. The remainder of this section will investigate the network performance under this scenario. Figure 6.5 shows a plot of

Table 6.2: Table showing the NN classification accuracy for different RS state ranges for a binary input of 0.5V. Note: The software classification accuracy is 97.1%.

RS Range (Ω)	Classification Accuracy (%)
5k-10k	98
5k-30k	99
5k-50k	86
5k-100k	46

Table 6.3: Table showing the NN classification accuracy for different RS state ranges for an input of 0.5V. Note: The software classification accuracy is 98.4%.

RS Range (Ω)	Classification Accuracy (%)
5k-10k	96.3
5k-30k	78.3
5k-50k	56.3
5k-100k	17.3

the NN classification accuracy when different percentage of memristors are dead. To investigate this, the memristors in the NN were first initialized as usual. A random number generator then output 10 numbers that corresponded to instances of memristors and the RS of these memristors were then set to 1GΩ and the performance of the network was then tested. The process of generating more instances of memristors and killing them was then repeated to obtain all the results.

From the graph, beyond 8% of dead memristors the network classification accuracy begins to significantly decrease as the absence of MAC operations become significant to cause classification errors. Until around 8%, the network classification accuracy remains relatively constant as some MAC operations are not vital for classification i.e the current contributions to the bitlines are insignificant. This phenomenon is commonly referred to as 'pruning' [42] in the field of machine learning where certain neurons and connections are eliminated, as shown in Figure 6.6, to compress the size of models in order to reduce inference time and to allow them to run on small low power devices. The iterative method carried out above using a random number generator is a naive method and more efficient methods are used in practice [42] that maximise eliminating neurons and connections while minimising the loss in performance. Nevertheless, this investigation also demonstrated that common software machine learning techniques can also be applied to selectorless memristor crossbar arrays.

6.3 Inference Frequency

This section investigates the classification accuracy of the memristor NN with inference frequency. The reader is reminded that the memristor neural network is asynchronous, it does not rely on a clock signal to pass data through the architecture and the neurons in the output layer output a voltage instantaneously when a voltage is applied on a wordline in the input layer. The propagation delay between a voltage applied on a wordline and the neurons in the output layer outputing a voltage in response was investigated in Figure 6.7. As designed in the Verilog-A control module, it takes 0.01ns for a voltage on the wordline to change. In response, it takes a neuron in the output layer around 0.01ns to also change output voltage. However, this

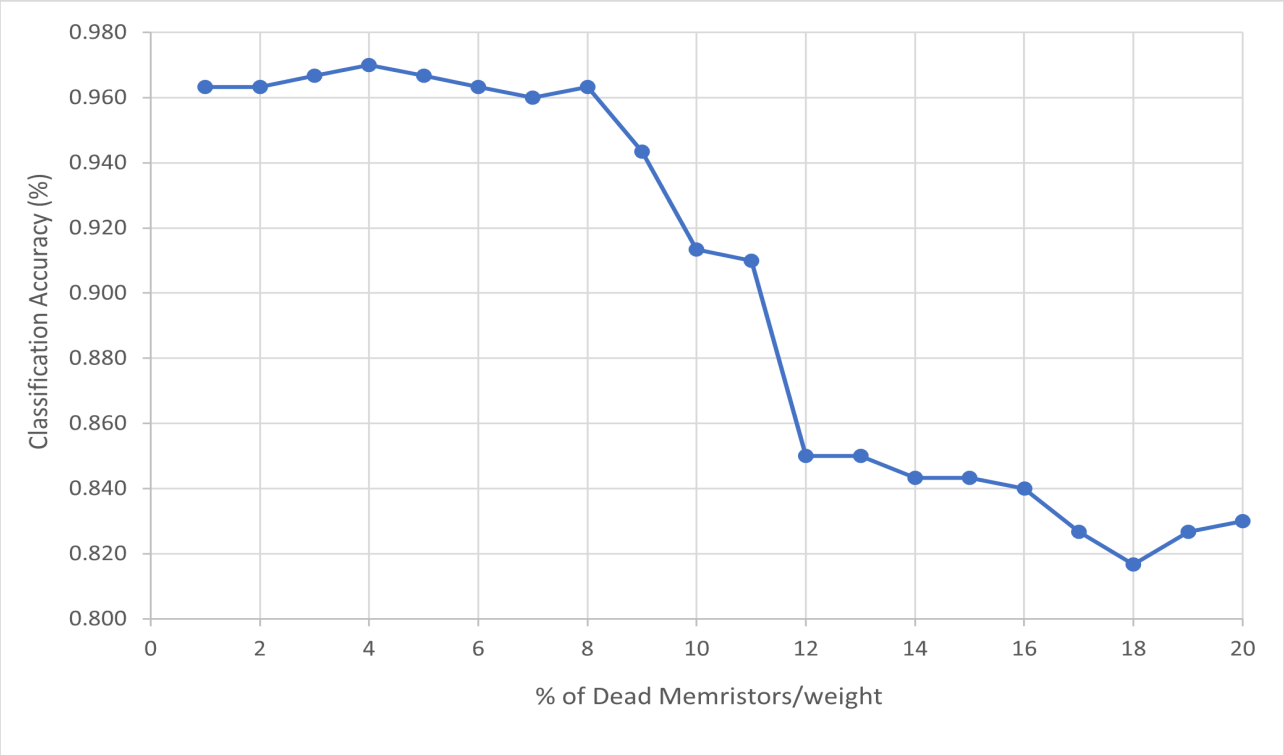


Figure 6.5: Graph showing the NN classification accuracy when different percentage of memristors are dead.

measurement is not accurate as the output voltage from the neurons continuously change due to sneak currents, so the marker was not accurately placed.

As mentioned in Section 5.2 the clock, load and sample signals respectively determine when a data sample is read from a text file, loaded on to the worlines and when the neurons in the output layer are sampled to determine the number detected. Thus, the system can be set to any clock frequency but the clock frequency will determine the pulse width and hence the time duration for which voltages corresponding to binary pixel inputs are applied on the wordlines. In other words, this is the inference frequency. As the clock frequency increases the rate at which data samples are loaded on to the wordlines increases.

Figure 6.8 shows a plot of the classification accuracy of the memristor NN at various inference/clock frequencies. The relationship is exponential, below 1MHz the classification accuracy rapidly decreases. This is because at low frequencies voltages are applied on the wordlines for a longer time (pulse width) resulting to more current flowing through the memristors causing greater perturbations from the initial RS the memristors were set at. The evidence of this can be seen by the exponentially decreasing blue curve which shows a particular memristors RS change after inference has been performed on the entire data set i.e (Initial RS - RS after 300 test data)/Initial RS. The results of this experiment are reliable since the the memristors were initialised to the same weights for each experiment and the same test data set was used.

The classification accuracy of a low frequency inference exercise can be restored to the same accuraccy as a high frequency inference exercise by refreshing the weights after every few test samples. Figure 6.9 shows the classification accuracy of the memristor NN as different number of

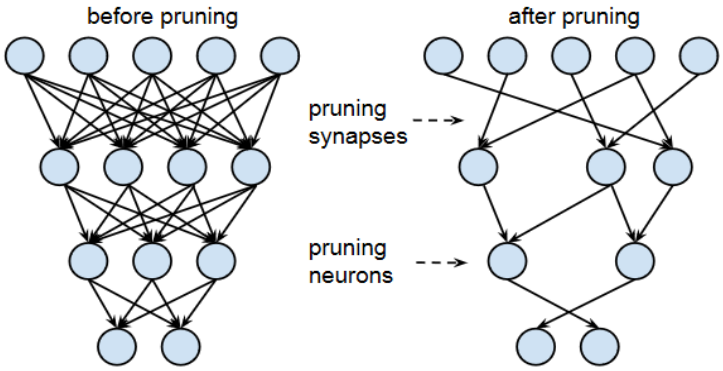


Figure 6.6: Visualising weight/neuron pruning.

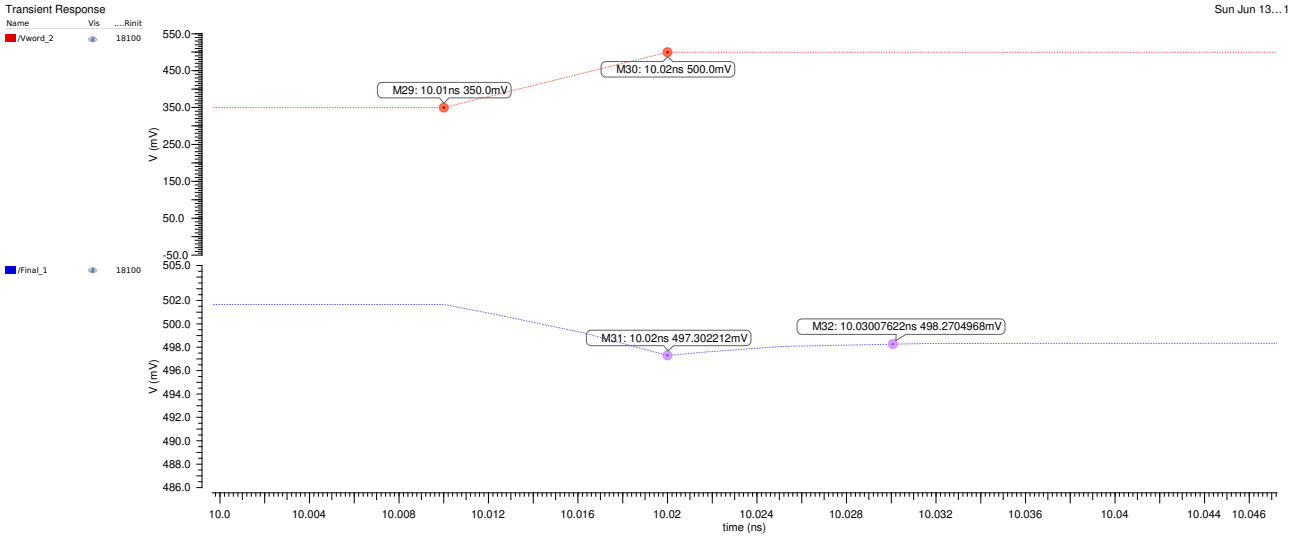


Figure 6.7: Graph showing a voltage applied on a wordline and an output from a neuron in response to the input.

test samples are passed to it before the weights are refreshed. From Figure 6.8, the classification accuracy when the entire data set is passed to the network running at 0.05MHz is 88%, however by refreshing the weights every 10 samples the classification accuracy can be increased to 97%.

6.4 Performance on different datasets

Out of curiosity, to test the inference capabilities of the memristor NN, the network was also tested on a data set consisting of 4 bit errors and 5 bit errors rather than 3 bit errors (which the software network was trained on and corresponding weights used). The results can be seen in Table 6.4. Since the classification accuracy exceeded 90% for all the tests it is fair to say that learning did indeed take place. The results are reliable since in all 3 tests the memristors in the NN were programmed to the same inital RS and the data sets were all based on binary pixel inputs of 0 or 0.5.

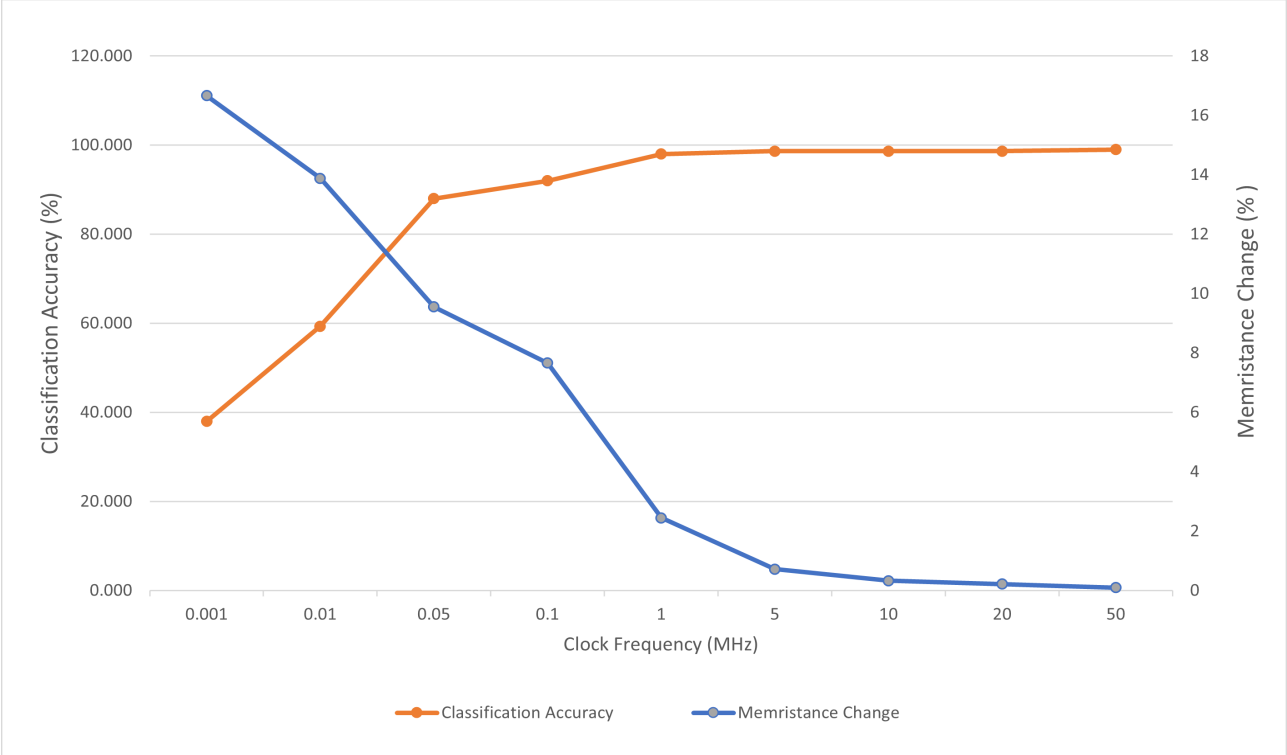


Figure 6.8: Graph showing the relationship between clock frequency and network accuracy and clock frequency and the change in the RS of a memristor.

Table 6.4: Table showing classification accuracies of the memristor NN on data sets of images with various BER.

BER	Memristor NN Classification Accuracy (%)	Software Classification Accuracy (%)
3	99	97
4	94.3	96.3
5	90.3	92.8

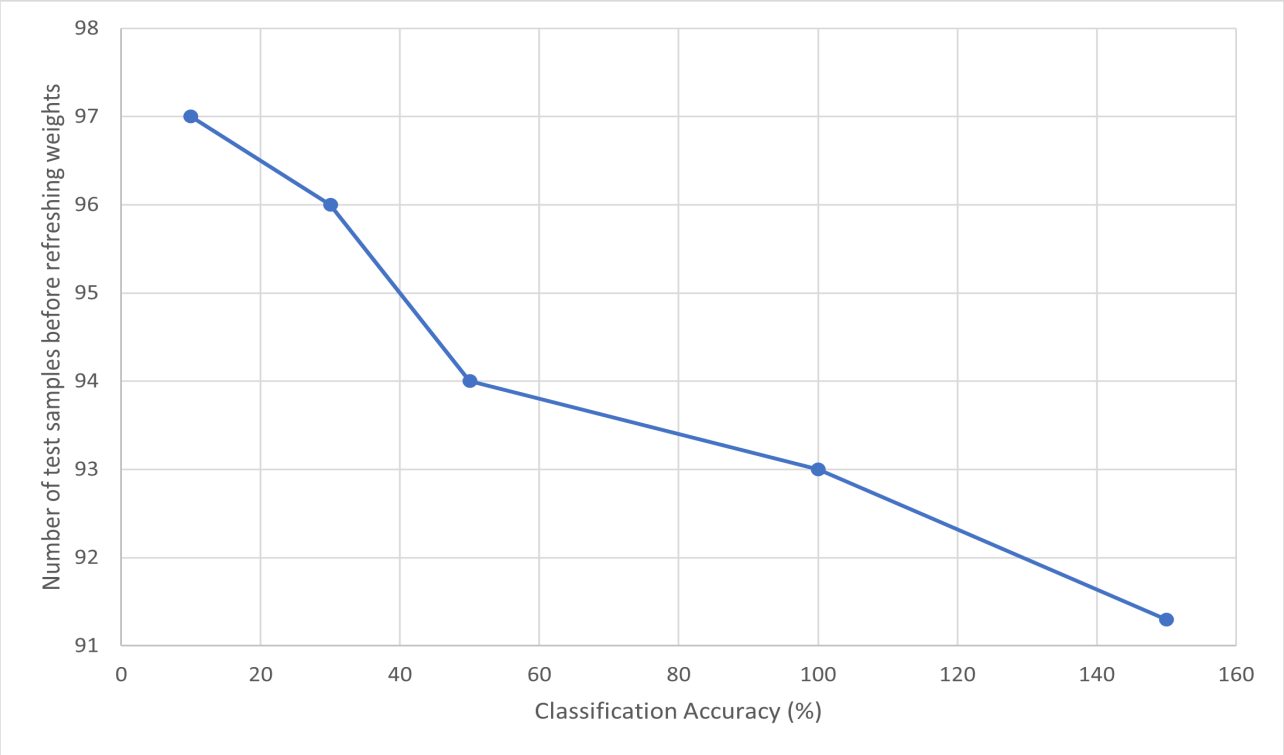

Number of test samples before refreshing weights
98
97
96
95
94
93
92
91
0
20
40
60
80
100
120
140
160
Classification Accuracy (%)

Figure 6.9

Chapter 7

Evaluation

The fear of creating something without a predictable behaviour has for years deterred researchers from investigating the potential of selectorless memristor crossbar arrays in implementing neural networks. This chapter critically reflects on the work conducted and evaluates its significance in the Neuromorphic Engineering and Hardware Acceleration industry.

The results presented in the previous chapter not only demonstrate that selectorless memristor crossbars are able to perform machine learning computations reliably, but in certain test cases outperform the inference classification accuracy of their software model. Since, the only difference between the memristor NN crossbar and the software model was the presence of sneak currents it gives grounds to the claim that an optimum propagation of sneak currents in a cossbar can further improve performance - perhaps explained by stochastic resonance. The greater performance of the '0 or X' in comparison to the '0.35 or 0.35+X' wordline write scheme furthermore supports this claim.

Additionally, the investigations carried out in the previous chapter lay the foundation for designing a new generation of neuromorphic devices that can outperform current designs. The results suggest, that by optimising the memristance RS range for any input the system can achieve roughly the same inference performance as a software model. In addition, if a system uses memristors that have a narrow RS range, the inputs to the system can be modulated to different values to optimise performance.

As the system is asynchronous in nature with a propagation delay of just around 0.01ns, it has a throughput that is dependent on the rate at which samples are passed to it. Cadence simulations were able to verify this upto 100MHz, beyond which there is no evidence to believe the system would not perform otherwise. From the exponential relationship between clock frequency and network classification accuracy, it is fair to postulate that at a gigahertz frequency the memristor neural network classification accuracy may equal the software model as sneak currents become irrelevant.

As of now, a true estimate of the performance can not be made as the ideal TIAs in the neurons would be replaced by transistors in an IC thus introducing more propagation delays. Moreover, depending on the frequency at which inference is run (data samples passed to crossbar) the weights would need to be refreshed after a certain number of samples have been passed to the system. Nevertheless, the in-memory computing nature of memristor NN will outperform traditional Von Neumann computing devices on latency since they only need to fetch the data

samples from memory and not the system parameters (weights).

The tests conducted further show that memristor NN crossbars are process tolerant, being able to retain their performance when upto 8% (for this model) of memristors are non-functional, a metric that will encourage funding to fabricate such devices.

Chapter 8

Further Work

Providing experimental evidence to verify that selectorless memeristor NN crossbars can reliably perform machine learning applications was just the start in realising the fabrication of an IC that can be reconfigured to implement any neural network. The investigations carried out on the network provide future designers with a set of guidelines however a lot more work needs to be done before an IC can be fabricated.

As the objective of this project was verifying that selectorless memritor crossbars can perform machine learning computations reliably the RS of the memristors were initialised to a particular value. However, if memristor crossbars are to be integrated as part of an IC that can be reconfigured to implement any machine learning function their RS must be programmable. Programming a discrete memristor to a particular RS is a challenge itself requiring a control algorithm to vary the write pulse amplitude and pulse width on the go. An even bigger challenge and perhaps impossible due to sneak currents is programming memristors in a selectorless crossbar array.

Nevertheless, it is believed that online-training; which is passing a data sample to the memristor NN, then learning (running an optimization model like back-propagation) and updating the RS of the memristors (by writing pulses) could allow the memristors in the crossbar array to converge towards the optimum weights/RS. The converged memristor NN crossbar could have an even higher performance than an equivalent software model as the presence of sneak currents could result to even better performance for reasons described previously.

To get a more accurate estimate of the propagation delay of the system, the TIAs need to be replaced with actual designs. In addition, a more thorough investigation on finding the optimum RS range for a particular input needs to be conducted and lastly, it would be good if all the results could be replicated for another larger dataset like the MNIST dataset to further reinforce all the findings in this project.

Chapter 9

Conclusion

This project has replicated a software model of a neural network using a selectorless memristor crossbar to provide experimental evidence to support their reliability in performing machine learning computations. Not only was the aim of the project achieved but, further investigations carried out led to deeper insights into the behaviour of these architectures and challenged the long held notion that sneak currents are detrimental to crossbar arrays. The work carried out additionally lays the foundation for creating the next generation of ultra-low power machine learning processors that will hopefully one day become incorporated in our everyday lives.

Looking back at this journey, besides strengthening my analog and digital IC design skills, this project has also allowed me to gain valuable experience working with industrial IC design tools and creating Verilog-A designs. More importantly, I have found a research area that I'm passionate about and would like to spend more time working on.

Bibliography

[1] Von Neumann, J. IEEE Ann. Hist. Comput. 15 (1993): 27. Web.

[2] S. A. McKee, in Proc. of the 1st Conf. on Computing Frontiers, ACM Press, Santa Clara, CA 2004.

[3] Wulf, Wm McKee, Sally. (1996). Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News. 23.

[4] Keckler, Stephen Dally, William Khailany, Brucek Garland, Michael Glasco, David. (2011). GPUs and the Future of Parallel Computing. Micro, IEEE. 31. 7 - 17. 10.1109/MM.2011.89.

[5] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[6] J. Dukovic et al., "Through-silicon-via technology for 3D integration," 2010 IEEE International Memory Workshop, Seoul, 2010, pp. 1-2, doi: 10.1109/IMW.2010.5488399.

[7] Mead, C. How we created neuromorphic engineering. Nat Electron 3, 434–435 (2020). https://doi.org/10.1038/s41928-020-0448-2

[8] Merolla PA, Arthur JV, Alvarez-Icaza R, et al. Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science. 2014;345(6197):668-673. doi:10.1126/science.1254642

[9] J. Tang, F. Yuan, X. Shen, Z. Wang, M. Rao, Y. He, Y. Sun, X. Li, W. Zhang, Y. Li, Adv. Mater. 2019, 31, 1902761.

[10] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," Nature, vol. 453, no. 7191, pp. 80–83, 2008.

[11] Yang, J., Strukov, D. and Stewart, D. Memristive devices for computing. Nature Nanotech 8, 13–24 (2013).

[12] A. Adeyemo, "Design and Analysis of Memristor Based Reliable Crossbar Architectures, PhD thesis," Oxford Brookes University, Oxford, UK., 2018.

[13] L. Chua. Memristor-The missing circuit element. IEEE Transactions on Circuit Theory, 18(5):507–519, 1971.

[14] L.O. Chua and Sung Mo Kang. Memristive devices and systems. Proceedings ofthe IEEE, 64(2):209–223, Feb 1976. ISSN 0018-9219. doi: 10.1109/PROC.1976.10092.

[15] I. H. Inoue, S. Yasuda, H. Akinaga, and H. Tagaki. Nonpolar resistance switching of metal/binary-transition-metal oxides/metal sandwiches: Homogeneous/inhomogeneous transition of current distribution. Physical Review B, 77:035105, 2008.

[16] J. S. Lee, S. Lee, and T. W. Noh. Resistive switching phenomena: A review of statistical physics approaches. Applied Physics Reviews, 2:031303, 2015.

[17] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. -, p. -, 2012.

[18] Cai, Weiran and Tetzlaff, Ronald. (2014). Synapse as a Memristor. 10.1007/978-3-319-02630-57.

[19] Serrano-Gotarredona T., Masquelier T., Linares-Barranco B. (2019) Spike-Timing-Dependent-Plasticity with Memristors. In: Chua L., Sirakoulis G., Adamatzky A. (eds) Handbook of Memristor Networks. Springer, Cham. https://doi.org/10.1007/978-3-319-76375

[20] Hebb, D.O.: The Organization of Behavior. A Neuropsychological Study. Wiley, New York (1949)

[21] Kandel, E.R.: In Search of Memory, 400. W. Norton and Company, New York (2006)

[22] Citri, A., Malenka, R. Synaptic Plasticity: Multiple Forms, Functions, and Mechanisms. Neuropsychopharmacol 33, 18–41 (2008). https://doi.org/10.1038/sj.npp.1301559

[23] Chua L. Memristor, Hodgkin-Huxley, and edge of chaos. Nanotechnology. 2013 Sep 27;24(38):383001. doi: 10.1088/0957-4484/24/38/383001. Epub 2013 Sep 2. PMID: 23999613.

[24] R. E. Uhrig, "Introduction to artificial neural networks," Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics, Orlando, FL, USA, 1995, pp. 33-37 vol.1, doi: 10.1109/IECON.1995.483329.

[25] Horowitz, M. 2014 IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC). 2014. Web.

[26] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks," Future Internet, vol. 12, no. 7, p. 113, Jul. 2020.

[27] https://www.marketresearchfuture.com/reports/hardware-acceleration-market-8249

[28] Trafton: Mimicking the brain, in silicon. http://news.mit.edu/2011/brain-chip-1115 (2011)

[29] Akopyan F., et al.: Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 34 1537–1557 (2015)

[30] J. Hutchby and M. Garner, "Assessment of the potential and maturity of selected emerging research memory technologies," International Technology Roadmap for Semiconductors, 2010.

[31] D. L. Lewis and H.-H. Lee, “Architectural evaluation of3D stacked RRAM caches,” in 3D System Integration, 2009. 3DIC 2009. IEEE International Conference on, 2009, pp. 1–4

[32] Liu, C., et al.: A spiking neuromorphic design with resistive crossbar. In: Design Automation Conference (DAC), pp. 14:1–14:6 (2015)

[33] Hassan A.M., Liu C., Yang C., (Helen) Li H., Chen Y. (2019) Designing Neuromorphic Computing Systems with Memristor Devices. In: Chua L., Sirakoulis G., Adamatzky A. (eds) Handbook of Memristor Networks. Springer, Cham.

[34] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, “Memristorbased memory: The sneak paths problem and solutions,” Microelectronics Journal, vol. 44, no. 2, pp. 176–183, 2013.

[35] I. Messaris, A. Serb, S. Stathopoulos, A. Khiat, S. Nikolaidis and T. Prodromakis, ”A Data-Driven Verilog-A ReRAM Model,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 12, pp. 3151-3162, Dec. 2018, doi: 10.1109/TCAD.2018.2791468.

[36] M. Douthwaite, F. García-Redondo, P. Georgiou and S. Das, ”A Time-Domain Current-Mode MAC Engine for Analogue Neural Networks in Flexible Electronics,” 2019 IEEE Biomedical Circuits and Systems Conference (BioCAS), Nara, Japan, 2019, pp. 1-4, doi: 10.1109/BIOCAS.2019.8919190.

[37] M. S. Qureshi, M. Pickett, F. Miao and J. P. Strachan, ”CMOS interface circuits for reading and writing memristor crossbar array,” 2011 IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, 2011, pp. 2954-2957, doi: 10.1109/ISCAS.2011.5938211.

[38] Soton Verilog A

[39] B. Li, Y. Wang, Y. Wang, Y. Chen and H. Yang, ”Training itself: Mixed-signal training acceleration for memristor-based neural network,” 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), Singapore, 2014, pp. 361-366, doi: 10.1109/ASP-DAC.2014.6742916.

[40] https://www.jeremyjordan.me/nn-learning-rate/

[41] Moss, F. (2004). Stochastic resonance and sensory information processing: A tutorial and review of application. Clinical Neurophysiology. 115. 267-281. 10.1016/j.clinph.2003.09.014.

[42] https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9

[43] Ren, HP., Bai, C., Baptista, M. et al. Weak connections form an infinite number of patterns in the brain. Sci Rep 7, 46472 (2017). https://doi.org/10.1038/srep46472

Appendix

A

Verilog-A code for memristor model

```
`include "constants.vams"
`include "disciplines.vams"

module analytical (p, n);
  inout p, n;
    electrical p, n;
  parameter real Ap = 743.47;
  parameter real tp = 6.51;
  parameter real An = -68000;
  parameter real tn = 0.31;
  parameter real kp = 5.11e-4;
  parameter real rp0 = 16710;
  parameter real rp1 = 0;
  parameter real kn = 1.17e-3;
  parameter real rn0 = 29300;
  parameter real rn1 = 23690;
  parameter real Rinit = 16250;
  parameter real eta = 1;
  parameter real ap=0.24;
  parameter real bp=2.81;
  parameter real an=0.24;
  parameter real bn=2.81;
    real Rmp;
    real Rmn;
    real vin;
    real RS;
    real IVp;
    real IVn;
    real IV;
    real first_iteration;
    real R0_last;
    real dt;
    real it;
    real svp;
    real svn;

    analog function integer stp;
      real arg; input arg;
      stp = (arg >= 0 ? 1 : 0 );
    endfunction


```

```
        analog begin
      if (first_iteration==0) begin
      it=0;
      R0_last=Rinit;
      end
      dt=$abstime-it;
      vin=V(p,n);
      Rmp=rp0+rp1*vin;
      Rmn=rn0+rn1*vin;

        if (vin>0)
        RS=(1/kp)*ln(exp(eta*kp*Rmp)+exp(-eta*kp*(Ap*(-1+exp(eta*tp*abs(vin))))*dt)*(exp(eta
    *kp*R0_last)-exp(eta*kp*Rmp)));
        else
        RS=-(1/kn)*ln(exp(-eta*kn*R0_last+eta*kn*(An*(-1+exp(tn*abs(vin))))*dt)-exp(-eta*kn*
    Rmn)*(-1+exp(eta*kn*(An*(-1+exp(tn*abs(vin))))*dt)));

      if (RS>=Rmp && vin>0)
      RS=R0_last;
      if (RS<=Rmn && vin<0)
      RS=R0_last;

      IVp=ap*(1/RS)*sinh(bp*vin);
      IVn=an*(1/RS)*sinh(bn*vin);
      IV=IVp*stp(vin)+IVn*stp(-vin);
      I(p, n)<+ IV;

      R0_last=RS;
      first_iteration=1;
      it=$abstime;

    end
endmodule

```

B

Python code used to generate data for a single number.

```
#clean 0
zero = [0,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,1,1,0]

#generate zeros with 3 bit errors
zero_tmp = np.zeros(shape = (200, 32))

for i in range(200):
  zero_tmp[i] = zero

  #generate random array index numbers
  random_index_a = random.randint(0,31)
  random_index_b = random.randint(0,31)
  random_index_c = random.randint(0,31)

  #print(random_index_a, random_index_b, random_index_c)

  if zero[random_index_a] == 1:
    zero_tmp[i][random_index_a] = 0
  else:
     zero_tmp[i][random_index_a] = 1

  if zero[random_index_b] == 1:
    zero_tmp[i][random_index_b] = 0
  else:
     zero_tmp[i][random_index_b] = 1

  if zero[random_index_c] == 1:
    zero_tmp[i][random_index_c] = 0
  else:
     zero_tmp[i][random_index_c] = 1

```

C

Verilog-A code used to mitigate the effect of sneak paths in a 3x3 crossbar

```
    analog
    begin
        @(initial_step)
        begin
            Wclk = (1/Fclk)/2;
            clkstate = -1;
            readState = -1;
            counter = 0;
        end

        @(timer(0, Wclk))
        begin
            clkstate = -clkstate;
            counter = counter + 0.5;
            readState = counter==readPulseno ? 1:-1;
        end

        //direct bitlines 1 and 2 to transimpedance amplifier by turn on pmos
        V(writeSet2) <+ -1.8;
        V(writeSet) <+ -1.8;

        //set all memristors to around 12.5KOhms
        V(voltageOut) <+ transition( (clkstate>0 && counter<2*writePulses)? write:0, 1n, 1n, 1n);
        V(voltageOut2) <+ transition( (clkstate>0 && counter<2*writePulses)? write:0, 1n, 1n, 1n);
        V(voltageOut3) <+ transition( (clkstate>0 && counter<2*writePulses)? write:0, 1n, 1n, 1n);

        //set target memristor to a diff RS by setting row 1 to Vw
        V(voltageOut) <+ transition( (clkstate>0 && counter<5*writePulses && counter>4*writePulses)? write:0, 1n, 1n, 1n);
        //hold other rows to Vw/2
        V(voltageOut2) <+ transition( (counter<5*writePulses && counter>4*writePulses)? half_write:0, 1n, 1n, 1n);
        V(voltageOut3) <+ transition( (counter<5*writePulses && counter>4*writePulses)? half_write:0, 1n, 1n, 1n);
        //direct target memristor bitline to ground by turning on NMOS
        V(writeSet3) <+ transition( (counter<5*writePulses && counter>4*writePulses)? 1.8:-1.8, 1n, 1n, 1n); //only during target MR write is nmos on otherwise pmos on

        //mitigate sneak current effect  on row 1 and 2
        V(voltageOut2) <+ transition( ( counter<9.5*writePulses && counter>5*writePulses)? -0.7:0, 1n, 1n, 1n);
        V(voltageOut3) <+ transition( ( counter>9.5*writePulses && counter<14.5*writePulses)? -0.7:0, 1n, 1n, 1n);


        //read RS
    //  if ($abstime >= 1.6m && $abstime <= 1.61m)
    //  begin
    //      V(voltageOut) <+ read;
    //      V(voltageOut2) <+ read;
    //      V(voltageOut3) <+ read;
    //  end

    end

endmodule
```

D

Verilog-A code that implements the tanh and sigmoid function activation functions.

```
`include "constants.vams"
`include "disciplines.vams"

module tanh_neuron(tanh_in, tanh_out);

input tanh_in;
output tanh_out;

electrical tanh_in;
electrical tanh_out;

real x;

  analog
  begin
    x = V(tanh_in);
    V(tanh_out) <+ tanh(x);
  end

endmodule

module sigmoid_neuron(sigmoid_in, sigmoid_out);

input sigmoid_in;
output sigmoid_out;

electrical sigmoid_in;
electrical sigmoid_out;

real x;

  analog
  begin
    x = V(sigmoid_in)-0.35;
    V(sigmoid_out) <+ 1/(1+exp(-x));
  end

endmodule


```