Memristive Crossbars: ALU Design, Testing, and Fault Analysis for Neuromorphic Applications

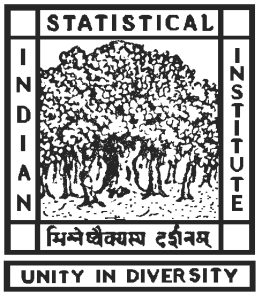
A thesis submitted for the degree of
Doctor of Philosophy

by

Manobendra Nath Mondal

Advisors

Prof. Susmita Sur-Kolay
and
Prof. Bhargab B. Bhattacharya



Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203 B. T. Road, Kolkata - 700108
West Bengal, India
July 2023

Dedicated to
To my family

Acknowledgements

The journey towards the completion of this thesis has been a formidable one, filled with both challenges and triumphs. Along this path, I have been fortunate to receive unwavering support from those around me, and I would like to take this opportunity to express my heartfelt gratitude to all those who have contributed to making this experience truly memorable.

First and foremost, I extend my sincere appreciation to my supervisors Prof Susmita Sur-Koley, and Prof. Bhargab B. Bhattacharya, whose constant guidance and suppport have been instrumental throughout my Ph.D. Prof. Bhattacharya's profound knowledge and invaluable insights have enriched my understanding of the subject matter , while Prof. Sur-Kolay's expertise in the field have made the complex research work more accessible. I am truly grateful for their unwavering encouragement, motivation, and constructive criticism, which have propelled me towards achieving my goals. I would also like to acknowledge the exceptional support of Prof. Nabanita Das, and all the faculty members from whom I have gained invaluable knowledge during this journey.

Every result presented in this thesis owes its accomplishment to the support and collaboration of fellow lab-mates and collaborators. I wholeheartedly appreciate the immense help and teachings of Sandip-da, Sukanta-da, and Sudip-da throughout these years. I would like to extend my sincere thanks to Manjari Pradhan, Animesh Basak Chowdhury, and Santlal Prajapati for their fruitful contributions to our co-authored paper.

I am also thankful for the support and camaraderie of my colleagues, including Debasis-da, Dibakar-da, Oishila-di, Samir, Abhishek, Tapalina, and Mukta. Their presence and encouragement have been invaluable. I extend special appreciation to Dulu-da for his caring presence throughout these days. My sincere thanks to Sachi-da and Debu-da for their technical assistance. I would also like to acknowledge to the non-teaching staff of the Advanced Computing and Microelectronics Unit, ISI for the constant assistance in official matters.

I would like to express my sincere gratitude to the Indian Statistical Institute, Kolkata, for providing the fellowship for my Ph.D. research. I would also like to thank the anonymous reviewers and experts who have contributed their valuable time and insights to the review process.

Words cannot convey my gratitude to Subrata-da who have been always beside me in

all my ups and downs through his unconditional love and support. My heartfelt thank goes to my friends Samir, Buro, Bishnu, Piu, Anirban, Dibyendy, Shouvik, Banani, Swadesh, Kaustav, Daribaba, Partha, Subho, Sadhan, Jitu, Panda, Pabitra, Raju-da, Subrata-da, Pitam for all the emotional support and caring they provided. Your friendship is irreplaceable.

Lastly, I offer my deepest and most heartfelt gratitude to my parents and all my family members. Without the unwavering support, boundless love, and constant encouragement from Borda, Mejda, and Didi, I would have never been able to include "Dr." before my name. I am deeply grateful to my wife, Suchetana, whose unwavering belief in me has been the cornerstone of my perseverance and achievements.

To all the individuals mentioned above, and to those whose names may have inadvertently been left out, I extend my sincere gratitude for the profound impact you have had on my academic and personal growth. It is through the collective support of these exceptional individuals that I stand proudly before you today, presenting this thesis.

Manobendra Nath Mondal

Abstract

In 1971, Professor Leon Chua introduced the notion of a memristor, the fourth fundamental passive circuit component alongside resistor, capacitor, and inductor. The resistance of this two-terminal device depends on the current through it; thereby a memristor is similar to a resistor with memory. In 2008, a group of researchers at HP Labs built the first memristor successfully and demonstrated its characteristic resistance-switching behavior. Its unique properties and compatibility with CMOS technology has made it a powerful circuit element and has significantly influenced design paradigms.

Recent developments have shown that memristors are promising for designing memory and logic subsystems, which can store multiple states of memory by utilizing the analog variation of resistance in the cells. By combining CMOS components with memristor cells, hybrid systems can be created where CMOS components can perform computation-in-memory (CIM), while memristor cells can store data in a non-volatile manner. Memristor-based crossbars (MBCs) realised as a 2D-array of memristors, have been particularly effective for performing certain types of computations, such as vector-matrix multiplication (VMM) and vector outer product, which are crucial in neuromorphic computing systems. Developing practical and reliable memristive crossbar-based systems for various applications still poses significant challenges which can hinder their performance and scalability. This thesis tackles several challenges head-on, offering innovative solutions that elevate their performance, reliability, and scalability.

In this thesis, we introduce novel designs for an Arithmetic Logic Unit (ALU) that utilize differential currents passing through a hybrid-memristor crossbar network. The ALU performs integer addition, subtraction, multiplication, and logical operations in the binary domain, using both analog and digital components. The analog component provides the peripheral control circuit, input voltages and logic values to the crossbar in the form of memristor-states. The variation of the analog output current is then sensed and converted back to discrete logic bits using A/D converters. Our simulation studies demonstrate that our proposed designs are more efficient than previous approaches, with lower memristor cost and computation-cycle time required for integer addition and multiplication.

Next, we envisage a 2D memristor crossbar as a network and identify certain paths that are suitable for fault sensitization. In order to optimize testing time for large scale full-size square and rectangular memristive crossbars, we propose a path-based technique

guided by maximum matching in bipartite graphs. In order to address the testing optimization problem in both complete and incomplete crossbars, we employ an Integer Linear Programming (ILP) formulation. By comparing the results of the ILP formulation with our proposed path-based techniques, we observe that our method minimizes the number of paths required. Through simulations conducted in LTspice, we validate the effectiveness and superiority of our approach over previous techniques in terms of test time and fault coverage.

Finally, we present a thorough analysis of the impact of various hard faults on memristive crossbars utilized in neuromorphic computing. This study is critical as comprehending the effects of such faults can enhance the reliability and efficacy of fault-tolerant memristor based neuromorphic computing systems with resource constraints. Moreover, understanding the effect of faults on individual layers of memristive crossbar-based neural architecture can guide the discovery of optimal architecture design to solve specific problems with predefined accuracy.

Overall, this thesis contributes to advancing memristor-based computing systems by addressing ALU design, fault sensitization, test time optimization, and the impact of faults on neuromorphic architectures. The findings provide valuable insights to improve the reliability and performance of future fault-tolerant memristor-based systems across a wide range of applications.

CONTENTS

LIST OF PUBLICATIONS RELATED TO THIS THESIS

The thesis is based on the following publications of the author.

In Journals

[1] **M. N. Mondal**, S. Sur-Kolay and B. B. Bhattacharya, "Test Optimization in Memristor Crossbars Based on Path Selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 294-307, Jan. 2023
doi: 10.1109/TCAD.2022.3168782

[2] **M. N. Mondal**, S. Sur-Kolay and B. B. Bhattacharya, "Scalable Design of Memristive ALU via Differential Current Sensing," to be submitted.

In Peer-reviewed International Conference Proceedings

[3] **M. N. Mondal**, S. Sur-Kolay and B. B. Bhattacharya,"Current Comparator-Based Reconfigurable Adder and Multiplier on Hybrid Memristive Crossbar," 2020 *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* 2020, pp. 494-499.
doi: 10.1109/ISVLSI49217.2020.000-8

[4] **M. N. Mondal**, S. Sur-Kolay and B. B. Bhattacharya,"Selective Sensitization of Useless Sneak-Paths for Test Optimization in Memristor-Arrays," 2019 32^{nd} International Conference on *VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID),* 2019, pp. 383-388. doi: 10.1109/VLSID.2019.00084

[5] **M. N. Mondal**, A. B. Chowdhury, S. Prajapati, S. Sur-Kolay and B. B. Bhattacharya, "Analysis of Fault Sensitivity in Memristive Neural Architectures," to be submitted.

LIST OF FIGURES

LIST OF TABLES

CHAPTER

ONE

INTRODUCTION AND THE SCOPE OF THE THESIS

Contents

1.1 Introduction

Memristors are fundamental passive circuit elements that possess the traits of both memory and resistors. Due to their unique properties and compatibility with CMOS technology, they

have emerged as powerful circuit elements that significantly influence upcoming analog and digital design paradigms. In the realm of memory subsystems, memristors have emerged as a highly promising technology. Leveraging the analog variation of current-induced resistance, a 2D-crossbar architecture constructed with memristor arrays offers an efficient platform for storing multi-valued memory states. Memristors' non-volatility, low power operation, high density, and high switching speed make them promising candidates for providing alternative solutions to conventional high-speed Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM) (RK21). Moreover, memristors do not have leakage power due to their non-volatile behavior, which makes them suitable candidates for Flash memories. Integration of CMOS components with non-CMOS memristor cells further enhances the scope of their applications in various complex system designs.

In recent times, memristors have shown significant promise in the context of a new generation of logic synthesis. Memristor-based physical architectures have the ability to store and process information on the same network, which supports a platform for enabling computation-in-memory (CIM). In the realm of computing, CIM holds significant promise, presenting a novel approach to data processing (LBL+18). However, its current performance trails behind that of traditional CMOS designs. Notably, in-memory computing excels in specific tasks characterized by low computational complexity but high demands for data movement (REL+22; AMY+23). As applications demand enhanced performance in in-memory computation, the variety of memristor technologies underscores the necessity for a standardized benchmarking procedure. This ensures fair evaluations across crucial metrics like multi-bit memory capacity, low-power operation, endurance, retention, and stability (SMK+19).

The memristor demonstrates synaptic behavior akin to that of biological synapses found in the human brain. Its features, such as non-volatility, the capacity to modify resistance based on voltage amplitude and duration, and continuous input/output characteristics, function similarly to biological synapses. A memristor-based crossbar (MBC) is highly scalable, with a vast number of memristive devices arranged in a two-dimensional grid. This configuration enables parallel processing of matrix-vector multiplication, vector outer product, and other operations with high efficiency and low energy consumption, which typically incur significant costs in terms of space, time, and energy. Additionally, the analog nature of memristors makes them ideal for performing the computationally demanding tasks required for neuromorphic computing. The inherent variability in memristor characteristics, including resistance states and switching dynamics, may introduce inaccuracies in compu-

tations and memory storage. However, in the context of neuromorphic computing, these non-ideal characteristics can be reframed as opportunities for innovation. Research indicates that embracing device non-ideality can benefit the development of neuromorphic systems. Strategies like non-ideality-aware training algorithms and random weight initialization, as explored in (LPL+21a; LPL+21b), illustrate that incorporating device imperfections into the computational framework can enhance performance and efficiency in neuromorphic computing applications. The approaches presented in (ZLB+22; SBK+22) signify a paradigm shift, leveraging inherent device characteristics to amplify the capabilities of neuromorphic systems, thereby demonstrating the potential for novel and more robust computing paradigms. This introduction section covers the definition, characteristics, modeling, and applications of memristors, as well as memristive crossbars, the motivation behind this thesis, and an outline of the thesis.

1.2 Memristor Model

Circuit theory considers four primary circuit variables, namely the current i, the voltage v, the charge q, and the flux-linkage ϕ. A passive two-terminal circuit element can be described in terms of the relationship between any two of these four parameters. For instance, a resistor is defined by the relationship between voltage v and current i, an inductor by the relationship between flux-linkage ϕ and current i, and a capacitor by the relationship between charge q and voltage v.

Chua demonstrated as in Figure 1.1 (Chu71) that there are six possible pairs of relations that can be represented by mathematical equations. Of these six equations, five are well-established and have been extensively used in circuit theory. These equations are:

- For a resistor: $v = R \cdot i$ by Ohm's law the voltage v across a resistor is linearly proportional to the current i flowing through it, where R is the resistance of the resistor.
- For a capacitor: $i = C \cdot \frac{dv}{dt}$ which relates the current i flowing through a capacitor to the rate of change of the voltage v across it, where C is the capacitance of the capacitor.
- For an inductor: $v = L \cdot \frac{di}{dt}$ which relates the voltage v across an inductor to the rate of change of the current i flowing through it, where L is the inductance of the inductor.
- $dq = idt$ relates the infinitesimal change in charge (dq) to the current (i) and the infinitesimal change in time (dt) through which the charge is flowing.

- By Faraday's law of electromagnetic induction, $v = \frac{d\phi}{dt}$, which relates the voltage v induced in a circuit to the rate of change of the magnetic flux ϕ linking the circuit.

The relation of the magnetic flux ϕ between two terminals of a circuit element as a function of the electric charge q passing through it was undefined until Chua proposed a mathematical model for a two-terminal passive circuit element called a memristor. Memristors exhibit unique properties that differ from the three traditional passive circuit elements, namely, resistors, capacitors, and inductors.

- For a memristor: $i = g(\phi)$ which relates the current i flowing through a memristor to the magnetic flux ϕ linking the circuit, where $g(\phi)$ is a non-linear function that depends on the history of the current and magnetic flux.

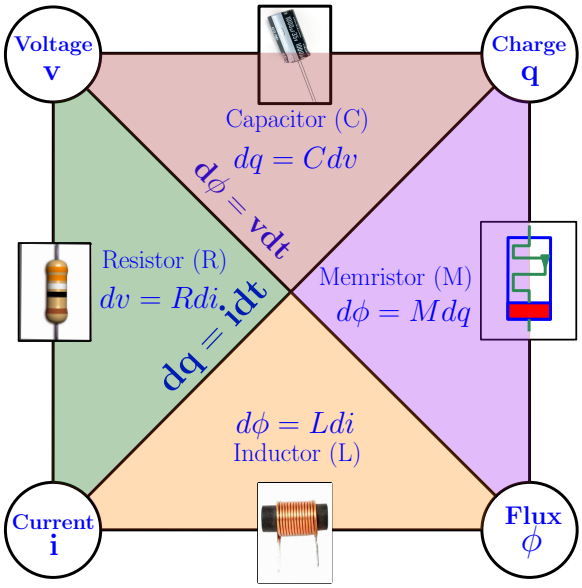


Figure 1.1: Relations among the four basic circuit elements in circuit theory (Chu71)

Before the discovery of Chua, there were experimental observations of pinched hysteresis loops in two-terminal electrical measurements in various material systems, which led to the development of devices based on those observations. In 1960, Bernard Widrow proposed a three-terminal device called a Memristor, where the device's memristance is the time integral of the current that passes through it. However, he did not claim it as a fundamental element in circuit theory. Chua later proved that it is impossible to construct an equivalent circuit for a memristor using only passive nonlinear resistors, capacitors, and

inductors. Furthermore, Chua proposed a rigorous mathematical model that can represent all types of memristors developed through different processes and materials.

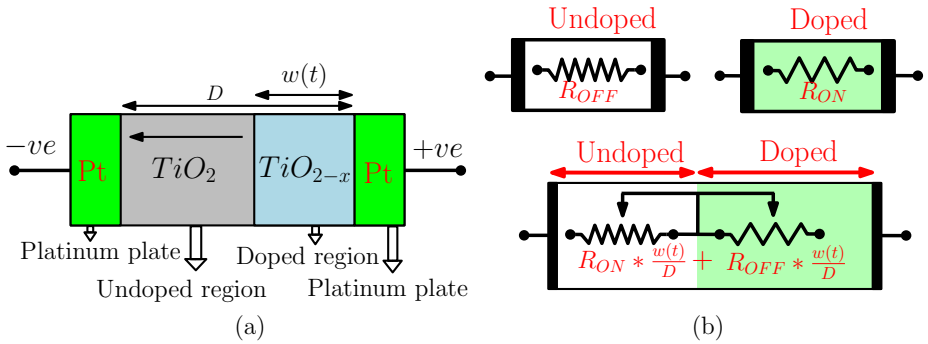


Figure 1.2: (a) Pt/TiO_2 memristor proposed by HP Labs., (b) its equivalent resistive model.

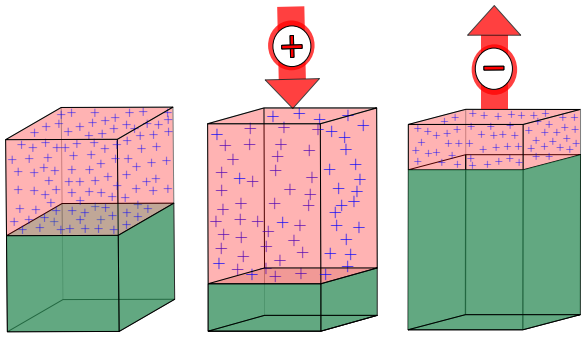


Figure 1.3: Change in length of doped region due to application of external bias voltage.

In 2008, researchers from HP Laboratory successfully produced the first memristor (DBSW08). They proposed a simple structure consisting of a thin semiconductor film (Tio_2 in 1.2(a)) of length D, sandwiched between two metal contacts (Platinum in 1.2(a)). A portion of the semiconductor film ($w(t)$ in 1.2(a)) is doped with a high concentration of dopants (by positive ions in 1.2(a)) which acts as a low resistance region R_{on}. The rest of the film has very low or zero dopant concentration and has a much higher resistance R_{off}. When an external bias voltage $v(t)$ is applied across the device's terminals, due to the drift of the charged dopants, the boundary between the two regions are shifted as shown in Figure 1.3. The device's total resistance is determined by R_{on} and R_{off}, where R_{on} and R_{off} are the device's resistance values when $w(t) = D$ and $w(t) = 0$, respectively. An equivalent resistive model of a partially doped memristor is shown in 1.2(b).

$$dv(t) = (R_{on}\frac{w(t)}{D} + R_{off}(1 - \frac{w(t)}{D}))i(t) \tag{1.1}$$

$$\frac{dw(t)}{dt} = \mu_v \frac{R_(on)}{D} i(t) \tag{1.2}$$

which yields the following formula for $w(t)$:

$$w(t) = \mu_v \frac{R_{on}}{D} q(t) \tag{1.3}$$

$$\tag{1.4}$$

By inserting equation (7) into equation (5) we obtain the memristance of this system, which for $R_{on} << R_{off}$ simplifies to:

$$M(q) = R_{off}(1 - \frac{\mu_v R_{on}}{D^2} q(t)) \tag{1.5}$$

1.3 Memristive Crossbar

A memristive crossbar is an architecture that comprises a 2D array of memristors with parallel horizontal metal wires, also known as Word Lines (WL), and parallel vertical metal wires, also known as Bit Lines (BL). At each crosspoint of a pair of WL and a BL is a memristor. Thus, a memristor, denoted by $M_{i,j}$, has one of its terminal connected to wire WL_i and the other connected to wire BL_j. As a result, the crossbar becomes memristive in nature.

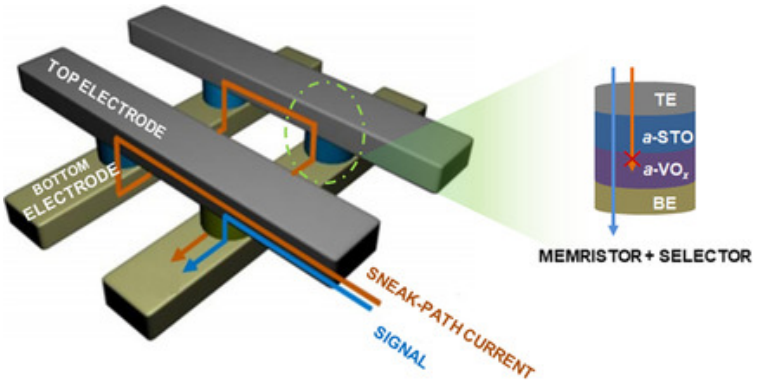


Figure 1.4: Schematic illustration of a crossbar memory array with normal and sneak current paths.

In order to read or write a conductance value of a memristor $M_{i,j}$, a pulse voltage is applied between wires WL_i and BL_j. But, while accessing a particular memristor in a crossbar architecture, along with the desired current (the blue solid line), a small amount of current may also flow through some un-selected memristor cells. This current is known as sneak-current (the orange solid line), and the path through which it flows is called a sneak-path as shown in Figure 1.4. Though sneak-path has a less critical impact on both machine learning and neuromorphic computing, but for sequential read-and-write of isolated memristors in crossbar arrays, the activation of sneak-paths is undesirable. Therefore, extensive research has been conducted to mitigate sneak-path leakage current in memristive crossbar architectures in functional mode. Techniques like adding an access element to a memristor ($1R$) cell to form composite cells such as one-transistor-one memristor ($1T1R$), one diode-one memristor ($1D1R$), one selector-one memristor ($1S1R$), self-rectifying memristors, etc., have been used to mitigate sneak paths (SZT+20; BS17; WRM+18; AT16).

1.4 Read and Write Operations in Memristive Crossbars

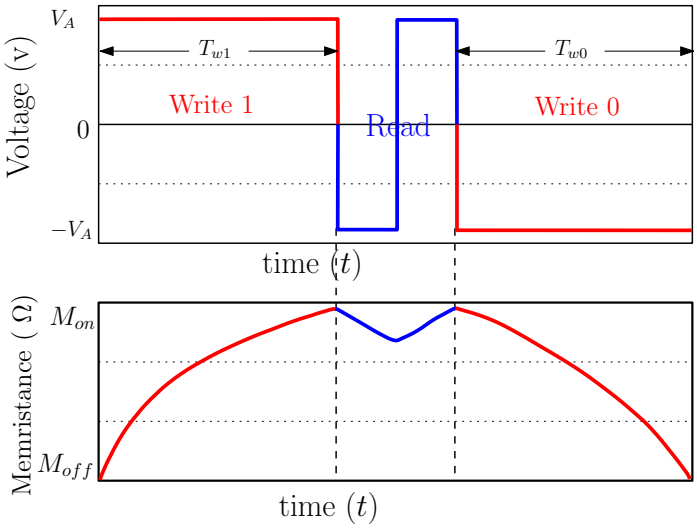


Figure 1.5: Read and write operations in a memristor

1.4.1 Read Operation

The read operation in a memristor is not as simple as the write operation. To read the internal state of a memristor, a voltage pulse with a small pulse width and amplitude V_A is applied across the memristor, and the output current is sensed to extract the internal information. However, the read voltage can perturb the memristor state. Therefore, if the read operation is not properly designed, the memristor state may be perturbed beyond its safety margin, resulting in soft errors after repeated reads. To prevent changes in the memristance of the memristor during the read operation, a two-stage read operation is used. The ideal read pattern consists of a negative pulse immediately followed by a positive pulse, with the magnitude and duration adjusted to create a zero net change in memristance, as shown in Figure 1.7. This ensures that the read operation does not affect the memristor state, and hence, the memristor can be repeatedly read without soft errors.

Another concern for the read operation is that the initial state of the memristor is not always guaranteed to be preserved after each read, especially when the memristor is in its peak high resistance state or peak low resistance state. For instance, if a memristor is stored at $w_0 = 0$, a negative step pulse during read operation would not affect its state, but a positive pulse would. The opposite is true when the memristor is in its opposite peak state. To address this issue, a threshold margin is defined while writing the memristor, and the read pulse width is limited such that the read process does not move the state beyond the threshold safety margin. This ensures that the memristor can restore its initial resistance value after a read cycle and prevents any undesired changes in its state.

1.4.2 Write Operation

To write into a memristor cell, one convenient method is to apply a voltage pulse with finite duration across the terminal of the memristor cell (XDJX11; HHL09). Suppose a memristor is initialized with a high resistance state, $w(0) = w_0 = 0$, and one desires to change it to a low resistance state, $w(t) = D$. In such cases, a positive voltage pulse with amplitude V_A and duration T_{w1} is required, as shown in Figure 1.5. The expression for the flux can be written as follows:

$$\phi(t) = \frac{\beta D^2}{2\mu_v}\left[(1 - \frac{w_0}{D})^2 - (1 - \frac{w(t)}{D})^2\right] \tag{1.6}$$

Where μ is the average drift mobility and β is the fit parameter. Therefore, for a fixed

V_A, the pulse width required to write a memristor from $w(t) = 0$ to $w(t) = D$ is given as

$$T_{w1,w0} \geq \frac{\phi_D}{|V_A|} \; where \; \phi_D = \frac{\beta D^2}{2\mu_v} \tag{1.7}$$

Accordingly, a pulse width of amplitude $-V_A$ with duration T_{w0} guarantee the memristor to reach from its low resistance state to high resistance state. Thus, a write signal that has duration equal or larger than Equation 1.7 can insure a successful write.

1.5 Motivation and Scope of this thesis

The memristor is a novel device with unique characteristics such as non-volatility, low power consumption, high density, and fast switching speed, make it an attractive choice for various applications, such as non-volatile memory design, digital and analog circuits, and neuromorphic computing, among others (Aka15; DK20; SHY18). Memristor-based Resistive Random Access Memory (RRAM) is considered one of the most significant applications for the next generation of memory technologies (VRK+09). RRAM has a similar structure to SRAM, but with the RS flip-flop replaced by a memristor as the basic memory element. It combines the advantages of current memories, such as SRAM's fast access time, DRAM's high-density organization, and Flash's non-volatile property. Several types of memristive crossbar-based memory subsystems have been proposed based on the characteristics of the memristor device (GKR19; GPJ+22).

One of the key advantages of memristors is their ability to be densely integrated into a crossbar configuration, which is known as the memristive crossbar. This makes it an ideal platform for various applications. Its compatibility with CMOS technology has further increased its appeal to researchers around the world.

In this thesis, we have addressed the challenges related to applications of memristive crossbars for in-memory computing and neuromorphic computing, in particular efficient design, testing and fault sensitivity analysis.

1.5.1 Roadblocks in Computation-In-Memory (CIM)

Conventional machines that use Von Neumann architectures (vN93), where memory and processing units are separated, often suffer from a bottleneck in processor-memory transactions. Furthermore, the CMOS implementation of such processors can be slowed down due to the limitations of power consumption. To address these issues, physical architectures

based on memristors have emerged as a promising candidate, thanks to their ability to store and process information on the same network (BSK+10). Memristors offer several advantageous features, including low power consumption and compatibility with CMOS technology. By using a "computation-in-memory" (CIM) scheme, memristors can help overcome the existing processor-memory bottleneck (CSAE19). Consequently, many academic and industrial research efforts are currently focused on logic implementation with memristors, and new design paradigms for CIM (KSW+14; KBL+14; BS10; TAS+14; TGY+17; GS17a; ARB+19) are continuously emerging. Memristive crossbars are particularly appealing in CIM architectures because they offer several potential benefits, such as high energy efficiency, fast computation, and compact implementation.

Developing practical and reliable computation-in-memory (CIM) architectures based on memristive crossbars poses several challenges that must be addressed. One such challenge is the need for effective algorithms and architectures that can leverage the unique properties of memristive crossbars for CIM applications. This includes efficient methods for performing arithmetic and logical operations using memristive crossbars, as well as methods for storing and accessing data in these systems. In this thesis, we propose a logic system based on differential currents to implement an adder on a hybrid-memristor crossbar network. We also describe how the adder modules can be repeatedly used to design a scalable arithmetic logic unit for various applications.

Current Comparator-Based Adder on Memristive Crossbar

First, a current comparator based re-configurable adder module is proposed on a memristor-based crossbar network. The module consists of two parts: a ripple carry adder (RCA) inspired addition and a carry-lookahead adder (CLA) inspired addition. The bit-parallel RCA module can be used to add k n-bit numbers in parallel, and the execution time for both modules is proportional to n. An Output Control Unit (OCU) comprising a simple CMOS current comparator followed by an A/D converter (ADC) analyze the output current. This eliminates the need for an expensive current sensing amplifier. The proposed designs are scalable and programmable according to the specifications and available crossbar area. The design also allows for the implementation of an multiple module together simultaneously on the same crossbar if available area is present.

Design of Multiplier and Logical Units on Memristive Crossbar

Next, a scalable implementation of in-memory subtractor, multiplier and logic units using a memristive crossbar that utilizes current sensing and analog peripheral circuits for computation in a mixed domain, is presented. The binary input is mapped as a state of a memristor in the crossbar topology, while the outcome of the arithmetic and logic operations appear in the analog domain. This strategy leads to reduced computation time and improved energy efficiency.

Both the subtractor and the shift-and-add multiplier deal with two unsigned n-bit binary numbers, and produce the output in $\mathcal{O}(n)$ cycles,. Logical operations on n-bit words such as NOT, AND, and OR are proposed. All these operations have attained reduction in memristor cost and computation-cycle time compared to previous approaches, as demonstrated by simulation studies.

1.5.2 Challenges in Testing Memristive Crossbars

Memristor cells suffer from non-deterministic nanoscale fabrication, despite their potential to revolutionize computing. Memristors are vulnerable to manufacturing defects and operational modalities (PFHE+21; HH11; SML21). Therefore, developing robust and efficient testing methods is crucial for detecting faults and ensuring the reliability of memristive crossbar-based systems (HKKC11; PAR15; XWG+21; JA22). In my research, I aim to address these challenges by developing effective testing techniques capable of detecting and diagnosing various types of faults, including stuck-at-faults, bridging faults, and open-faults. However, testing memristive crossbars presents several challenges, including developing test patterns and methods that can handle the inherent variability and noise present in memristive crossbars, while also being efficient and cost-effective for large-scale systems. Overcoming these challenges will pave the way for the commercialization of memristor devices, and significantly enhance their performance and reliability.

In this thesis, two novel test optimization techniques for producing test vectors for a 2D memristor-array are proposed. The first technique employs a graph-based path selection approach for optimizing tests in memristive square and rectangular crossbars. The second technique optimizes tests in memristive crossbars of any shape, including incomplete crossbars, through the use of an integer linear program (ILP). Our significant contributions are summarized below.

Test Optimization in Complete Memristive Crossbars

This study introduces a novel technique for testing the entire crossbar by accessing a single memristor and analyzing the sensitization of paths. The method converts the memristor network into an equivalent graph and utilizes path-covering algorithms to optimize the test time for a general class of faults. The algorithm applies maximum matching and Eulerian path algorithms to select paths in the graph that meet specific lower and upper bounds on path length. These bounds are determined by the electrical parameters of memristors and the resolution of the current-sense amplifier required for fault detection.

Not all sets of matchings provide a valid set of paths for optimal testing. Therefore, the properties of a Latin square are employed to choose eligible sets of matchings that ensure a valid path set. The proposed graph-based approach offers a minimum number of paths and surpasses all previous methods in terms of test cost and fault coverage.

Test Optimization in Incomplete Memristive Crossbars

Previous studies have proposed memristive crossbar-based hardware designs with missing memristors distributed randomly in the crossbar, also known as incomplete crossbars. While graph-based path-covering techniques have been efficient, they may not always be appropriate for incomplete crossbars. Therefore, this study proposes an Integer Linear Program (ILP) formulation for optimal path covering that can handle both full and incomplete crossbars uniformly. By comparing the results obtained from ILP, the proposed graph-based method offers a minimum number of paths (equivalent to the ILP solution) and surpasses all previous approaches in terms of test cost and fault coverage.

1.5.3 Fault-Tolerant Memristive Neural Networks: Key Challenges

The development of energy-efficient and scalable computing systems is a promising direction for memristive neural architectures (YBT+22). These architectures employ memristive devices that can modify their resistance based on the current that flows through them. This property is ideal for neural networks since it allows information to be stored and retrieved in a way that mimics the functionality of neurons in the brain. In particular, memristor-based crossbars (MBC) have proven highly effective in computing operations such as matrix-vector multiplication and vector outer product, which are typically costly in terms of space, time, and energy. Consequently, a significant number of artificial neural networks (ANNs), recursive neural networks (RNNs), and spiking neural networks (SNNs) have been proposed using

MBCs (WCGW20; WSYZ15; BPC15; ZRCMPC+11; KASC+11). Compared to traditional CMOS-based accelerators, MBC-based accelerators have significantly less power consumption per computation due to their massive parallel computing capacity and low-latency data movement capabilities.

The inherent fault tolerance of neural networks is largely attributed to their commonly used activation functions, pooling layers, and ranking-based outputs, which are generally insensitive to computing variations (THG17; Piu01). Despite this, there are various obstacles associated with the development and evaluation of fault-tolerant memristive neural architectures. One significant challenge is the presence of hard faults, as well as the high variability and instability of memristive devices, which can result in unpredictable behavior and degrade the performance of the neural network (YBT+22).

As the last contribution of this thesis, we have thoroughly examined how SA0 and SA1 faults impact the accuracy degradation of both a memristive deep neural network (MDNN) and the LeNet 5 (LBBH98) network. To simulate faults, we trained both the networks with MNIST (Den12), and Fashion MNIST (XRV17) datasets, and converted them into memristive networks. Our analysis focused on the sensitivity of each layer to the SAFs, allowing us to determine the impact of each fault on network performance. These fault analyses are essential for ensuring the reliable operation of memristive neuromorphic computing systems and resource-constrained fault-tolerant system design.

1.6 Organization of the Thesis

The chapters of this thesis are as follows:

- Chapter 1 covers the basic structure, working principles, and read-write mechanisms of memristors.
- Chapter 2 provides a review of related works. It discusses various types of memristor models and crossbar architectures, as well as their applications in different areas.
- Chapter 3 presents two kinds of adder modules in a hybrid memristive crossbar for computation-in-memory (CIM).
- Chapter 4 describes memristive modules for subtraction, multiplication, and logic operations utilizing differentiable current, for a scalable design of an ALU.
- Chapter 5 proposes a novel technique of test optimization based on path selection in memristive square and rectangular crossbars.
- Chapter 6 focuses on test optimization in an incomplete memristive crossbar and

proposes an ILP based approach.

- Chapter 7 covers fault analysis in a memristive crossbar-based neural circuit and presents a fault-tolerant design.
- Chapter 8 summarises the contributions of this thesis and discusses potential directions for future research.

CHAPTER

TWO

REVIEW OF RELATED WORKS

Contents

2.1 Introduction

Incorporating both memristor technology and CMOS, a new generation of hybrid memory-array has emerged. This architecture employs memristors as a storage element and CMOS to construct interface circuits, which shows promise in alleviating the memory-processor bottleneck in Von Neumann architectures (HCJ18; YNX+18; BAC18). Moreover, researchers have proposed logic synthesis by activating sneak-paths in memristor crossbar-arrays (CJ17a). Today, memristors have found applications in memory systems, digital and analog circuits, neuromorphic computers, hardware implementation of deep neural networks, AI-accelerators, and other domains (ZM17; CTSC20). In this chapter, we provide a brief overview of various memristor models, memristive crossbars and their security implications, along with previous research works relevant to this thesis.

2.2 Memristor Models

Memristors have several distinctive attributes, including good scalability, low power consumption, flexibility, and compatibility with CMOS. Different types of memristor models have been proposed based on their application-specific characteristics. In this section, we discuss some of the major memristor models.

2.2.1 Linear ion-drift model

In the linear ion drift mode (JW08), the total width of the memristor D is shared into two regions. One region with a width of w (which serves as the system's state variable) has

a high concentration of dopants and thus a higher conductance. The second region has a width of $D-w$ and is an oxide region with low conductance. It is assumed that memristors have ohmic conductance, linear ion drift in a uniform field, and equal average ion mobility μ_v. As a result, the state equation for the state variable is:

$$\frac{dw}{dt} = \mu_v \frac{R_{on}}{D} i(t) \tag{2.1}$$

$$v(t) = \left(R_{on} \frac{w(t)}{D} + R_{off} \left(1 - \frac{w(t)}{D} \right) \right) \tag{2.2}$$

Here, R_{on} is the resistance of the device at $w(t) = D$, and R_{off} is the resistance of the device at $w(t) = 0$. The state variable $w(t)$ is bounded within the interval limits of $[0, D]$. To prevent $w(t)$ from exceeding the physical device size, a window function is multiplied by the derivative.

2.2.2 Non-linear ion-drift model

The experimental studies of the fabricated memristors have proved that the behaviour of the implemented memristors are quite different from the linear drift model and are non-linear in nature which leads to the development of the non linear ion drift model (BBB09). The current voltage relationship is described as;

$$i(t) = w(t)^n \beta \sinh(\alpha v(t)) + \chi[exp(\gamma v(t)) - 1] \tag{2.3}$$

where α, β, γ, χ are experimental fittings parameters. the differentiation of the state variable can be expressed as,

$$\frac{dw}{dt} = a.f(w).v(t)^m \tag{2.4}$$

where $f(w)$ is the window function. The memristor model is well fitted for designing logic gates.

2.2.3 Simmons tunnel barrier model

The Simmons Tunnel Barrier Model accounts for non-linearity and anti-symmetric switching characteristics (AFC15). Unlike the linear ion drift model, this model includes a resistor in series with the electron tunnel barrier. The state variable in this model is the width of the

Simmons tunnel barrier, denoted by x. The changes in x are exponentially dependent on the movements of the ionized dopants, as expressed below.

$$\frac{dw(t)}{dt} = C_{off}\sinh(\frac{i}{i_{off}}exp\left[-exp(\frac{x-a_{off}}{w_c} - \frac{|i|}{b})\frac{x}{w_c}\right], i > 0 \tag{2.5}$$

$$C_{on}\sinh(\frac{i}{i_{on}}exp\left[-exp(\frac{x-a_{on}}{w_c} - \frac{|i|}{b})\frac{x}{w_c}\right], i < 0 \tag{2.6}$$

where b, con, c_{off}, i_{on}, i_{off}, a_{on}, a_{off} are called as fitting parameters. The current-voltage relation for this model is written as

$$v(t) = \left[R_{on} + \frac{R_{off} - R_{on}}{x_{off} - x_{on}}(x - x_{on})\right] i(t) \tag{2.7}$$

2.2.4 Yakopcic neuromorphic memristor model

Yakopcic proposed a memristor model that incorporates several fitting parameters to offer greater flexibility in device structure (YTS+11). The state variable's derivative in this model is defined as the product of composite functions. One of these functions determines the programming threshold voltage, which includes constants for instant phase transitions. The other function is the window function.

This device model is particularly useful for neuromorphic computations. When subjected to a linear DC sweep input, the resulting spikes more closely resemble neural spikes than those obtained from a sinusoidal input.

2.2.5 TEAM – ThrEshold Adaptive Memristor model.

Many previously published memristor models had a complex and implicit current-voltage relationship. In 2013, the Threshold Adaptive Memristor Model (TEAM) was proposed to overcome this limitation to a large extent (KFKW13). This model extends the Simmons Tunnel Barrier Model by assuming that the state variable does not change below the threshold. The defined polynomial is a function of the device's internal state derivative and current. This assumption simplifies the model and improves its computational efficiency.

2.2.6 VTEAM – Voltage ThrEshold Adaptive Memristor model.

In many applications, the threshold voltage is more desirable than the threshold current. To address this, a new model called the Voltage Threshold Adaptive Memristor Model

(VTEAM) was proposed (KRFK15). Instead of using the threshold current, this model uses the threshold voltage. Despite this difference, both the TEAM and VTEAM models provide similar features. Both models are extensions of the Simmons Tunnel Barrier Model and assume that the state variable does not change below the threshold.

2.2.7 Stanford-PKU model

The Stanford-PKU RRAM Model simplifies the complex process of ion and vacancy migration into the growth of a single dominant filament, while preserving the essential switching physics (HAMC19). The device's resistance is defined by the tunneling gap, which is the distance between the tip of the filament and the opposite electrode. The current conduction is exponentially dependent on the tunneling gap distance, which is affected by the electric field, temperature-enhanced oxygen ion migration, and local temperature due to Joule heating.

2.3 Memristive Crossbars

The sneak-path problem is a significant challenge in memristive crossbars, which occurs when unintended current flows through unintended paths, causing errors in the output. To address this issue, researchers have proposed various memristive crossbar architectures.

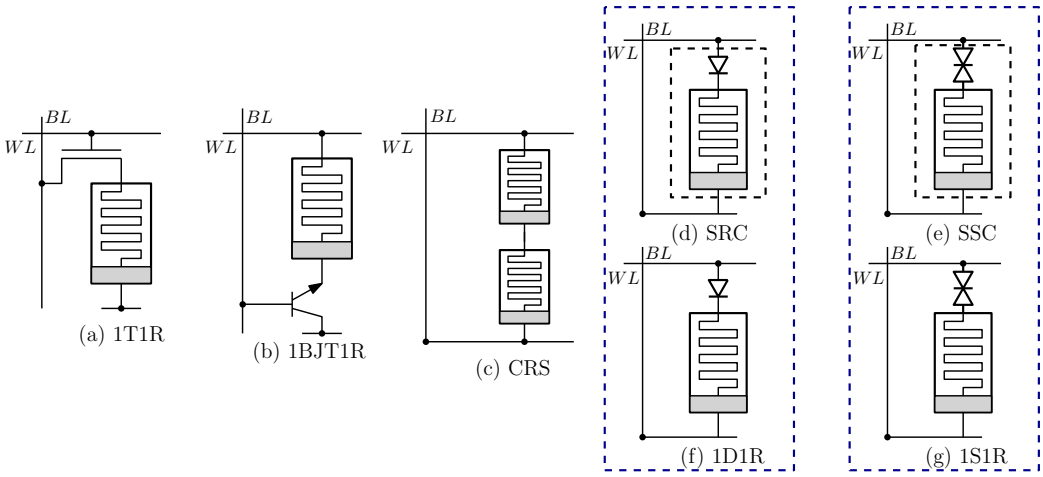


Figure 2.1: Seven types of possible solutions to solve the sneak-path current issue (a) $1T1R$ (b) $1BJT1R$ (c) CRS (d) $1D1R$ (e) $1S1R$ (f) SRC and (g) SSC, respectively.

2.3.1 1T1R cell and crossbar array

The $1T1M$ cell structure, as illustrated in Figure 2.1(a), utilizes a transistor connected in series with each memristor via RRAM/CMOS heterogeneous fabrication technology (LLY+14). In a 1T1R crossbar architecture, a specific cell is accessed by selecting the corresponding word line (WL), bit line (BL), and select line (SL) connected to the gate terminal of that cell. Therefore, this crossbar architecture is the most commonly used for computing-in-memory (CIM) applications. In the simple 1T1R, one entire row or column is selected at a time, while in the unfolded crossbar architecture, an individual cell can be precisely accessed by applying appropriate voltages to the corresponding WL, BL, and SL (MRHW10).

2.3.2 1S1R cell and crossbar array

A $1S1R$ cell is formed by connecting a selector in series with a memristor, as illustrated in Figure 2.1(g). The $1S1R$ architecture is one of the most preferred choices for $3D$ integration of RRAM-based crossbars since the selector is compatible with the memristor in terms of operating current and voltage ranges during both read and write operations (BS14). Since the selector is essentially a bidirectional highly nonlinear resistor, it has promising potential for high-density integration with RRAM for various applications.

2.3.3 1D1R cell and crossbar array

The $1D1R$ crossbar architecture, illustrated in figure 2.1(f), consists of a diode connected in series with a memristor. To address the sneak-path current issue, as recommended by the International Technology Roadmap for Semiconductors (ITRS), a combination of diode and transistor with a resistor in a single chip is more effective (CKS+10). The main advantage of $1D1R$ over $1T1R$ is its smaller area requirement and ease of fabrication.

2.3.4 1BJT1R cell and crossbar array

Obtaining a large current drivability is a challenging task in $1T1R$ or $1D1R$ cell structures. To address this issue, a new logic-compatible structure has been proposed, where a bipolar junction transistor (BJT) is vertically stacked underneath the memristor as depicted in Figure 2.1(b). The $1BJT1R$ structure provides high-performance current driving capability due to the high gain of the BJT. It can efficiently operate at a low voltage of $2V$ for the reset process and $1.5V$ for the set process (WTL+10).

2.3.5 CRS memory cell and crossbar array

An effective approach to address the sneak current problem in memristive crossbars is to utilize *CRC* cells, which consist of two bipolar memristors connected anti-serially, as shown in Figure 2.1(c). In this configuration, one memristor is set to a low-resistance state (LRS) while the other is in a high-resistance state (HRS), and a series resistor is added for stable operation (PW20). *CRC* cells can be categorized into two groups based on the polarity of the connection:

- *CRC* cells using two symmetric memory cells.
- *CRC* cells using two asymmetric memory cells.

This approach effectively reduces the sneak current in memristive crossbars, leading to improved reliability and accuracy.

2.3.6 SRC and crossbar array

Additional devices used to mitigate sneak path (selector, diode, transistor, etc.) bring advantages but also increase the read/write voltage, degrade memory stability, and affect scaling. Self-rectifying resistive memory (*SRC*) is an alternative solution. It is a metal-insulator-insulator-metal (MIIM) or MIM structure with a large work function difference between the top and bottom electrodes enabling the rectifying feature (YHZ+19).

2.3.7 SSC and crossbar array

A self-selective resistive switching approach can be used to overcome the issue of sneak path current. This approach involves integrating two oxide layers as an insulating layer in a single cell, which enables it to exhibit selective functionality (LXY+17).

2.3.8 Comparison of various architectures

A comparative analysis of various crossbar architectures is presented in Table 2.1, highlighting their individual advantages and disadvantages. The choice of a specific crossbar architecture depends on the intended application. Parameters such as on current, on/off ratio, Vset/Vreset, polarity, operating temperature, retention, and endurance are compared in the table.

Table 2.1: Comparison of few important features of different types of memristor based crossbar architectures.

Types	On current Amp. $[A]$	On/Off ratio	Operation polarity	Operation temperature $[t]$	Retentions Sec. $[s]$	Endurance	Ref.
$1S1R$	5×10^{-4}	10^9	Bipolar	–	–	10^6	(BS14)
$1T1R$	10^{-3}	10^8	Unipolar	300	10^5	10^8	(BS14)
$1D1R$	$\approx 10^{-4}$	10^8	Unipolar	473	$\approx 10^5$	10^4	(CKS+10)
$1BJT1R$	10^{-5}	≈ 10	Unipolar	–	10^3	10^5	(WTL+10)
CRS	10^{-2}	10^2-10^3	Bipolar	≈ 360	10^4	$\approx 2 \times 10^2$	(PW20)
SRC	10^{-4}	$\approx 10^4$	Unipolar	573	$\approx 2 \times 10^5$	$\approx 10^2$	(YHZ+19)
SSC	10^{-4}	10^{10}	Unipolar	450	10^6	10^6	(LXY+17)

2.4 Security Implications of Memristor-based Systems

ReRAM possesses inherent physical attributes that enhance content protection against hacking attacks and increase resistance to reverse engineering. The absence of charges makes it challenging to sense or alter its internal state using electron beams. Additionally, ReRAM can effectively withstand magnetic attacks due to its immunity to electromagnetic fields. The deep embedding of the ReRAM bit cell between two metal layers further enhances its resilience to optical attacks. However, a significant drawback in contemporary memristor computing systems, particularly in the era of artificial intelligence (AI), is the susceptibility of well-trained neural network (NN) models to theft threats when loaded onto these systems, particularly in edge devices. Advanced attacks, such as learning attacks and side-channel analysis, pose a substantial risk, potentially leading to the unauthorized acquisition of valuable well-trained NN models, as outlined briefly next for completeness' sake.

2.4.1 Black-box attacks and countermeasures

In specific memristor computing setups, attackers might not have access to the training dataset or the network weights. Nevertheless, they maintain the capacity to influence inputs, monitor outputs, and leverage side-channel analysis as integral components of their attack tactics. These attacks are commonly referred to as black-box models. In order to counter such threats, two primary approaches can be implemented for prevention.

One effective method for stealing neural network (NN) models is through learning attacks. In this technique, the adversary interacts with the system, querying it and observing

the corresponding outputs. After accumulating an adequate number of input/output pairs, a functionally similar NN model can be trained using this dataset (CCSZ23). In order to counter such attacks, the approach in (YLL+20) involved increasing voltage amplitude, accelerating memristor device obsolescence, and leading to a substantial reduction in computing system inference accuracy after a few queries. The authors in (RRP+23)) suggested harnessing the inherent stochasticity as a defense against learning attacks.

Side-channel attacks represent a category of non-invasive strategies wherein the targeted device is treated as a sealed black box. These techniques leverage various side channels, including power consumption, latency, and electromagnetism, to unveil sensitive information. Several studies, (HZS18; BBJP19; YFT20a), have delved into the theft of NN models using side-channel attacks. These incursions are directed at either traditional computing architectures or FPGAs. In the study by (YFT20b), a similar layer-by-layer processing approach was employed in the design of systems. The authors proposed using techniques to conceal memory access patterns for addressing such potential threats.

2.4.2 White-box attacks and countermeasures

In the white-box threat model, potential vulnerabilities arise as attackers may read the NN weights stored in the memristor crossbar, exploiting the non-volatility of memristor devices (AYS+19; ZHX19). Additionally, adversaries could employ micro-probing techniques to reverse-engineer the NN weight matrices stored in the memristor crossbar (HPJ+20).

Weight encryption methods involve straightforward encryption and decryption of NN weights during usage, necessitating additional write operations to memristor devices. The authors in (LCLL21) accelerated and optimized these processes to mitigate latency and energy consumption, while (CCT+19) aimed at minimizing required encryption/decryption data.

In the study cited ((ZZC+22)), NN weight transformation protection methods are employed, storing secured NN weights directly in computing units. This eliminates the necessity for weight rewriting operations, reducing energy and latency concerns and providing a distinct advantage over model encryption methods, as the weights remain consistently protected.

Approaches outlined in (AHSP23; JLZ+18) that rely on fingerprint embedding leverage the hardware variation of computing systems as a distinctive fingerprint, which is then incorporated into the NN weights. The pilfered NN model is rendered ineffective without this embedded fingerprint, as it is both hardware-dependent and challenging to replicate.

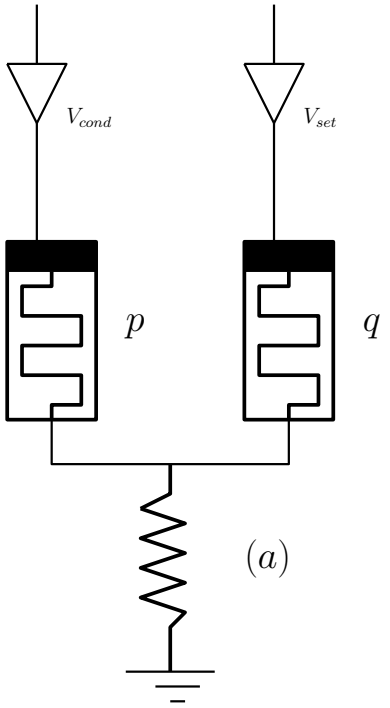
2.5 Memristive ALU Design for Computation-In-Memory (CIM)

With the exponential growth of data generated from commercial and social transactions, electronic devices, sensors, and scientific experiments, high-speed computation has become a crucial necessity for processing large volumes of data and extracting valuable insights. Traditional Von Neumann architectures (vN93) are hindered by processor-memory transaction bottlenecks, and CMOS-based processors, limited by power consumption, may further constrain their speed. These challenges have spurred research in the area of "computation-in-memory" (CIM), which is supported by a few promising technologies. Physical architectures based on memristors (VPC09) have the unique ability to store and process information on the same network (BSK+10). Memristors have emerged as a potential solution to overcome the existing processor-memory bottleneck due to their low-power consumption, compatibility with CMOS technology, and other supporting features (CSAE19).

In order to enable digital logic using memristors, the output logic can be determined by either the memristance or the output voltage level of the output memristors. The hybrid architecture of CMOS integrated with memristive networks provides a potential platform for implementing Boolean logic (SL12), where memristors serve as reconfigurable switches. Memristor ratioed logic (MRL)(TAAA16) employs a hybrid memristor-CMOS architecture to perform AND, OR, and XOR operations. Fuzzy logic has also been implemented using a similar technique(KS11). Bickerstaff and Swartzlander investigated the basic strategies for implementing ripple-carry adders and array multipliers with memristor-based structures for both analog and digital implementation (BS14). Memristor-based logic gained popularity with the implementation of material implication (IMPLY) logic and memristor-aided logic (MAGIC) (KSW+14; KBL+14), as all Boolean operations can be performed with either of these two gates.

The memristor-based IMPLY gate uses resistance to represent logical states, where high and low resistance correspond to logical zero and one, respectively. In this gate, resistor R_G ($R_{ON} < R_G < R_{OFF}$) is connected to two memristors, P and Q, which act as digital switches. The initial memristances of P and Q, denoted by p and q, respectively, serve as the inputs to the gate, and the output of the gate is the final memristance of Q. During operation, the memristance of both memristors changes. The circuit diagram is shown in Figure 2.2.

The IMPLY gate has a few limitations, such as the need for sequential voltage activation, which destroys the input state of the output memristor as the final output is stored



$case$	p	q	$p->q$
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

(b)

Figure 2.2: (a) An IMPLY logic gate operates as follows: the initial states of memristors p and q serve as inputs to the logic gate, while the final state of memristor q is the output, resulting from the application of voltages V_{SET} and V_{COND}. Both memristors are connected to a load resistor R_G. (b) The truth table of the IMPLY function (KSW+14).

in that memristor. Moreover, it requires complex control circuitry and consumes significant power. To address these challenges, a simple structure was proposed in (KBL+14), called MAGIC that executes the operation by applying a single voltage pulse at the circuit's gateway. In the MAGIC gate, a separate memristor is used for the output, and the initial logical state of the input memristors is the gate's input, while the final logical state of the memristor is the output, as illustrated in Figure 2.3.

Subsequently, many techniques have been developed for designing arithmetic structures based on IMPLY and MAGIC (BS10; TAS+14; TGY+17; GS17a). For example, an adder design with a memristor-based crossbar is described in (TGY+17) (RS18), and with memristor ratioed logic (MRL) in (ARB+19). A memristor-based shift-and-add multiplier was proposed in (GS17a)(GS17b), and a skeleton-based synthesis flow for Computation-in-Memory architectures was introduced in(YNA+20). Additionally, a supergate-aided (SAID)

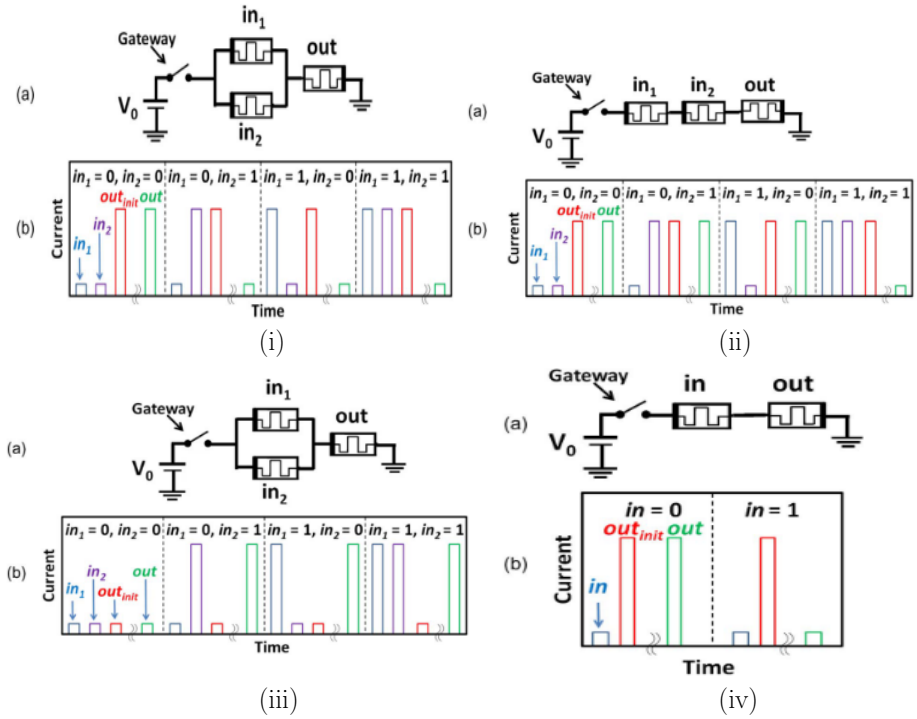


Figure 2.3: (i).(a) Schematic of a two-input MAGIC NOR gate; (i).(b) Simulations for all input combinations; (ii).(a) Schematic of a two-input MAGIC NAND gate; (ii).(b) Simulations for all input combinations; (iii).(a) Schematic of a two-input MAGIC OR gate; (iii).(b) Simulations for all input combinations; (iv).(a) Schematic of a two-input MAGIC NOT gate; and (iv).(b) Simulations for all input combinations (KBL+14).

logic synthesis approach tailored to MAGIC crossbars was presented in (TRB+19), and an exact synthesis method that utilizes IMPLY as its only logic primitives was proposed in (CC20). In (PZP21), a new synthesis method for implication logic circuits based on memristors was introduced, which provided a generalized rule for deriving the sequence of operations needed to realize any logic function written in the classical AND-OR form. However, these methods suffer from scalability issues and offer opportunities for further optimization of cost and computation time.

In order to tackle the challenges mentioned above, various functional synthesis techniques using data structures such as Binary Decision Diagrams (BDDs)(CJ17b; HCJ18), And-Inverter Graphs (AIGs)(BTP13), and Majority-Inverter Graphs (MIGs)(SSGD16) have been proposed for optimizing RRAM-based circuits. These techniques include free binary

decision diagram (FBDD)-based synthesis for CIM with memristive networks(HCJ18), mapping large Boolean circuits to memristor-based crossbars (XNT+18), and designing redundant binary adders with multi-valued memristor states for canonical signed digital (CSD) systems (EFR15). In (KUH19), a framework for automated synthesis of crossbar designs that implement approximate Free Binary Decision Diagrams (AFBDDs) using flow-based computing is proposed. Logic synthesis with memristors that require fewer operational pulses has also been studied previously (WLW+18). Numerous research endeavors focus on the design of analog-based arithmetic circuits to tackle the scalability challenges associated with memristor-based in-memory computing (KMNS21; AWY+23; MSR+20; QPMW21). In (FSD19), a fully automated logic synthesis approach for sequential circuits on hybrid CMOS-ReRAM architectures based on graph representations (i.e., BDDs and AIGs) is proposed. Apart from theoretical advancements, the literature reports a detailed review of experimental demonstrations of in-memory computations on a decent scale using memristor arrays for optimization and various machine learning applications (BZL+22; ZJL+18).

2.5.1 Gaps addressed in our work

The utilization of a Memristor-CMOS hybrid architecture is gaining popularity for in-memory arithmetic and modular designs to achieve adders and multipliers. This approach has proven beneficial in various fields such as AI/ML, edge-computing, and image compression due to its ability to emulate artificial neural networks (ANN) and function as hardware accelerators. The earlier ALU designs incorporating memristors faced challenges of multi-step operations and interference, thereby necessitating numerous time steps for execution and demanding a substantial chip area to implement a standard-size arithmetic logic unit. Given the inherently modular nature of digital designs, it is crucial to construct fundamental computing circuit modules and subsequently cascade them to create larger, complex units. Specifically, the design approach involves crafting arithmetic and logical circuits on a hybrid crossbar network with memristors, utilizing current sensing and incorporating analog peripheral circuits.

2.6 Testing of Memristive Crossbars

Memristor crossbars are susceptible to various manufacturing and field defects, which can affect their functionality. Therefore, robust testing of memristive crossbars-based systems is necessary to ensure their correctness. Various testing techniques have been proposed and

implemented in literature to evaluate the performance of memristor crossbars (GPM09b; HKKC11; HH11; HAS14; KKKS14; PAR15; LZW20; SML21; XWG+21; JA22).

In this regard, an electrical model of a MIM-based ReRAM memory element was presented in (GPM09b), and a robustness assessment was carried out in the presence of actual defects. Testing methods based on traditional March algorithms (BA00) were observed to be suitable for memristor-based memory arrays. In (CSW+15), the authors proposed the *Over-Forming (OF)* defect and the *Read-One-Disturb (R1D)* fault, which are specific to RRAM, and then a March algorithm called March C^* for detecting stuck-at-faults (SAF), read-one-disturb ($R1D$), and coupling faults (CF) in addition to the conventional RAM faults. However, the test time may increase significantly for large-sized arrays in this scheme.

In (HTH15), a *design-for-test (DfT)* scheme was introduced for memristor arrays. The authors presented a fault modeling approach for open defects using electrical simulation and suggested test methods for RRAMs. Their analysis revealed that in addition to some conventional memory faults, unique faults can also occur. They argued that traditional march tests may not be sufficient to detect such unique faults. Therefore, the authors proposed a new DfT concept that leverages the access time duration and supply voltage level of RRAM cells to improve the detection of these unique faults. Simulation results demonstrated that by making minor modifications to the circuit, the fault coverage was increased.

Kannan et al. proposed a fault model for SAF in memristive crossbar architectures by analyzing various physical defects. They also presented a test technique that utilizes sneak-paths in their previous work (KRKS13b). In a later study (KKKS15), the fault model was extended to include multi-valued cells, and the authors performed the test by reading multiple memristor-cells simultaneously, referred to as *region-of-detection (RoD)* as depicted in Figure 2.4. The size and shape of the RoDs are dependent on the nature of faults, and at least one memristor from each RoD is required to be accessed for testing as described in their earlier works (KRKS13b; KKKS15).

In (CL15), along with existing faults, two new faults, namely, a write disturbance fault (WDF) and dynamic write disturbance fault ($dWDF$), are defined. The WDF occurs when a write operation disturbs the stored data in adjacent cells, leading to errors in the stored data. The dWDF, on the other hand, is a more complex fault that occurs when the stored data in a cell is disturbed during a write operation and results in the disturbance propagating to adjacent cells during a subsequent read operation. In addition, a March test is proposed to cover the defined faults.

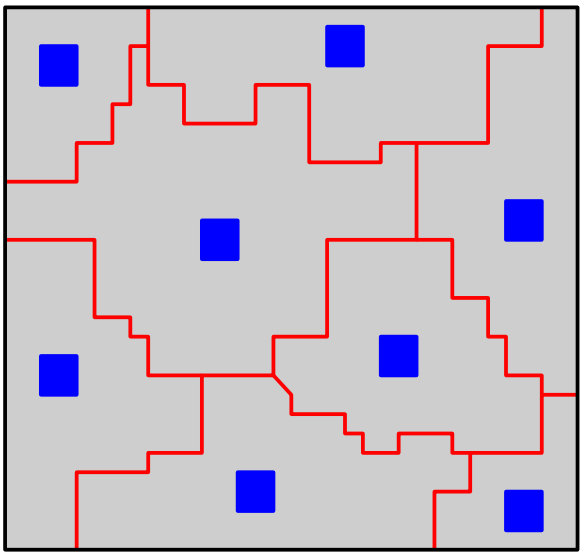


Figure 2.4: Test locations and coverage for an SA1 fault with sneak paths (not shown); the red lines demarcate the *RoD*, a dark blue square in a *RoD* is the memory element being sensitised and the gray region has other memory elements in its RoD whose faults can also be sensed at the output (KKKS15).

The work in (ZCX+16) introduced a testing technique that utilizes inherent sneak-paths in the 3D stacked one-transistor-N-RRAM (1TNR) structure. This method significantly improved the test time. In (MTH17), quick write operations were performed, and the impact of sneak paths was eliminated by applying appropriate voltages at corresponding WL and BL. Another work (LBJ+17) applied voltage bias to create various sneak-path distributions and used them to determine the exact location of each faulty memristor within three write-read operations.

In (XLN+17), a fault-tolerant online training method that includes two phases: fault detection and fault-tolerant training, was presented. The quiescent-voltage comparison method is utilized in the fault detection phase. In (SZZM18b), the author suggested a parallel testing technique based on memristive stateful logic that replaces the traditional March algorithm, leading to significant test time improvement. A parallel March-like test algorithm was presented for the CMOS Molecular (CMOL) architecture in (LWY+18), which covers defined faults caused by electrical defects, resulting in reduced test time compared to previous algorithms for the same architecture.

In (EVR19), the robustness of a chaotic circuit network is analyzed, assuming the existence of defective memristors (Stuck-at-OFF) arranged in a crossbar geometry. This study presents a test methodology that employs sneak paths for efficient test generation and establishes the relationship between additional test vectors and improved fault coverage. Another paper (JA22) proposes a test generation methodology that uses sneak-paths and describes the relationship between added test vectors and improved fault coverage for efficient test generation.

2.6.1 Gaps addressed in our work

Previous works use march or parallel march tests to detect faults in memristive crossbar architecture. However, these approaches become impractical for large or dense memories, as the testing time complexity becomes $\mathcal{O}(n^2)$ for an $n \times n$ crossbar. Certain suggested methodologies involve testing small clusters of memristor cells, leading to a reduction in test time despite maintaining a time complexity of $\mathcal{O}(n^2)$. A limitation arises as these methods necessitate access to at least one memristor from each cluster, which may not always be feasible. Additionally, these approaches do not address multiple $SA0$ or $SA1$ faults. Additionally, previous approaches focus on testing full memristor arrays with regular structures, while some crossbars may have missing memristors scattered randomly (incomplete crossbars (XNT+18; KDRB16)). Considering various hardware architectures that utilize incomplete crossbars, developing a universal crossbar testing technique becomes crucial to address all crossbar types and potential manufacturing faults.

2.7 Fault Analysis in Memristive Neuromorphic Architectures

Numerous artificial neural networks (ANNs) have been proposed using MBC technology, including recursive neural networks (RNNs) and spiking neural networks (SNNs)(WCGW20; WSYZ15; BPC15; ZRCMPC+11; KASC+11). MBCs offer significant advantages over traditional CMOS-based accelerators in terms of parallel computing capacity and low latency data movement, resulting in less power consumption per computation. However, the CMOS-memristor heterojunction fabrication process makes MBCs prone to faults, including permanent hard faults that occur during manufacturing and cannot be corrected, as well as soft faults resulting from inaccurate writing of memristor cells during data processing(DFR+15). As a result, the accuracy of MBC-based computing systems is limited and cannot reach their maximum achievable value (PMBH+15).

A lot of studies have investigated the implementation of spiking neural networks (SNNs) using MBC technology and spike-timing-dependent plasticity (STDP) to mimic biological neuroscience results (ZRCMPC+11; QBG11; NOBT12; CL12; LBSGCM+11; QBDG13; SGMP+13). However, guaranteeing the convergence of STDP-based learning for general purpose inputs is challenging (LNM05). MBC-based single-layer neural networks (SNNs) have also been proposed for small-scale problems, but they are not capable of handling large datasets (CZQK11; MRR12; SKM+13). Subsequently, MBC has been applied in complex artificial neural network hardware designs capable of handling large datasets to solve complex problems, including MNNS (HTY17; HGL+18a; YZS19; SSS+20), LSTM (ASKJ18; WWY+21), CNN (YAT17; WWZH18), pulse coupled neural networks (PCNNs)(XWZH18), and computing systems(HGL+18b).

While deep neural networks (DNNs) are often considered fault-tolerant due to their network connections and other features, this characteristic does not guarantee tolerance against hardware faults or prevent significant accuracy variations (LCX+22). Memristor devices, which are heterogeneous and high-density, are susceptible to manufacturing defects and process variations, leading to faulty crossbars that can significantly impact the classification accuracy of ANNs (CLC19).

A lot of research works have been conducted to detect, locate, and recover from faults in neural circuits. Paper (HFNT19) proposes systematic testing of computation-in-memory (CIM) architectures designed for bit-wise logical operations. In (XA19), two safety design techniques, Algorithm Based Atomic Error Checking-1 (ABAEC-1) and ABAEC-2, are presented for a Weight Stationary (WS) Convolutional Neural Network (CNN), along with fault diagnosis. Paper (LHSL17) proposes a defect-rescuing design that uses retraining and remapping algorithms to identify significant weights. In (ZUFE20), a framework that uses matrix transformations to handle stuck-at-faults and enable robust inference of deep neural networks (DNNs) is presented. Fault-tolerant training algorithms for DNNs are proposed in (LWJ+19; WXY+20), while paper (XCG+20) proposes a fault tolerance-aware hierarchical clustering method for ex-situ training. Finally, papers (CC21b; CC21a; CC22) propose the misclassification-driven training (MDT) algorithm to efficiently identify critical faults (FCFs) in the crossbar. In (SFP+23; SSB+22), the researchers have developed a specialized fault injection framework for logic-in memory (LiM) systems. This framework enables the simulation of various hardware faults and facilitates the evaluation of their impact on system behavior by introducing faults at different levels within the LiM architecture. The findings indicate that specific fault types, such as stuck-at faults or coupling faults, have a

significant detrimental effect on the performance of LiM-based neuromorphic systems.

2.7.1 Gaps addressed in our work

Many studies have shown that faults can affect classification accuracy differently depending on their type and location. However, the impact of different types of faults on different weight values at the same position is not well understood, and designing fault-tolerant algorithms without prior knowledge of the faulty cell's location is a promising area of research. Investigating the sensitivity of SAFs across various layers of a neural network and devising fault-tolerant mapping algorithms could be an intriguing avenue of research to address these concerns.

CHAPTER

THREE

DESIGN OF MEMRISTIVE ADDERS USING DIFFERENTIAL CURRENT SENSING

Contents

3.1 Introduction

An enormous amount of data is generated nowadays by various electronics gadgets, in addition to that from complex scientific experiments. High-speed computation is an essential requirement to process such large volume of data and to extract information embedded therein. Von Neumann architectures (vN93) are known to suffer from processor-memory transaction bottleneck, and CMOS-implementation of such processors being confronted with the power wall, may further impede their speed. These drawbacks have opened up a wide area of research concerning computation-in-memory (CIM) supported by a few promising technologies. Memristors have shown significant promise both in logic synthesis and memory-subsystem design. A 2D-crossbar of memristors can be used to store multi-valued memory states by utilizing the analog variation of current-induced resistance through memristor cells. The integration of CMOS components with memristor cells can further widen their design space for handling various complex systems. A crossbar built with CMOS-memristor hybridization supports a potential platform that enables computation-in-memory (CIM).

In this chapter we propose a logic system based on differential currents for implementing ADDER on a hybrid-memristor crossbar network. Simulation studies demonstrate that the proposed design reduces both memristor cost and computation-cycle time compared to previous approaches.

3.2 Motivation

Let us consider the addition of k unsigned numbers in parallel, each comprising n-bits. Considering bit-wise addition, the sum bit will be 1, if k is odd, and 0 otherwise; furthermore, at most $\lfloor \log k \rfloor$ carry bits are generated. We perform binary addition in a memristive network, where we map input bit 0 (1) to a high (low) resistance path. The output bit is determined by sensing the total output current that is determined by the number of low-resistance paths activated through the network.

Motivating Example: Consider a resistive model of a two-bit binary adder that adds two binary numbers, say $A = a_1a_0 = 11$ and $B = b_1b_0 = 01$ as shown in Figure 3.1. Each resistive block contains a pair of resistors having resistance R_{on} and R_{off} and at a

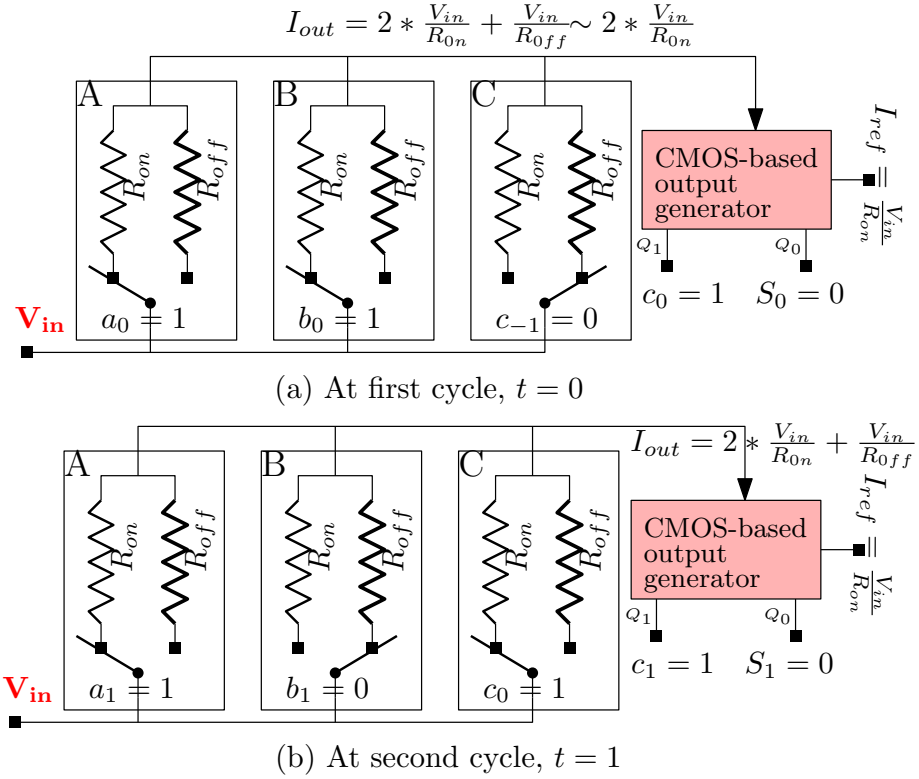


Figure 3.1: Input connections of the resistive circuit for a 2-bit adder at (a) $t = 0$ and (b) $t = 1$ to add two 2-bit binary numbers A and B.

time only one resistor is connected to the source voltage. Therefore, for designing a 2-bit adder, three resistive blocks A, B, and C (two blocks for the two input bits and one for the carry bit) are required. If the input bit is 1 (0) the source voltage is connected to R_{on} (R_{off}) in the corresponding block. When source voltage V_{in} is applied, the current passing through R_{on} or R_{off} is equal to $I_{on} = \frac{V_{in}}{R_{on}}$ and $I_{off} = \frac{V_{in}}{R_{off}}$, respectively. Usually, $I_{on} >> I_{off}$ ($\frac{R_{off}}{R_{on}} \geq 10^4$), and hence, I_{off} can be ignored. In order to process the output current, a CMOS-based circuit is used which identifies the number of low resistance paths (l) by measuring the total output current and then converts l into equivalent binary bits. As the maximum value of l can be three, the output needs two pins $Q = Q_1Q_0$, where Q_0 is the LSB and Q_1 is the MSB, respectively. In this setup, the output bit Q_0 represents the sum bit and Q_1 corresponds to the next stage carry bit.

The addition is completed in two cycles. At $t = 0$ (Figure 3.1(a)), the LSBs ($a_0 = 1$

and $b_0 = 1$) of the two input numbers are applied to the circuit, i.e., in both blocks A and B, R_{on} is connected to the voltage source. The initial carry being zero, R_{off} of C is connected to V_{in}. Total active blocks being two, the output is $Q = Q_1Q_0 = 10$ where the sum bit $Q_0 = S_0 = 0$ and next stage carry $Q_1 = c_0 = 1$. Similarly, at $t = 1$, the output is $Q = 10$ where $S_1 = 0$ and $c_1 = 1$ as shown in Figure 3.1(b). Therefore, the final result comprises two sum bits and the outgoing carry bit; thus the output becomes $Q = c_1S_1S_0 = 100$. Tables 3.1 and Table 3.2 depict the details of output-generation logic and mapping of input bits to the resistive network, respectively.

Table 3.1: Output generation logic for binary adder

$I_{ref} = V_{in}/R_{on}$, $l = \lfloor I_{out}/I_{ref} \rfloor$			
Output current I_{out}	#Low-resistance path (l)	Output logic	
		Q_1	Q_0
$3 * V_{in}/R_{off}$	0	0	0
$V_{in}/R_{on} + 2 * V_{in}/R_{off}$	1	0	1
$2 * V_{in}/R_{on} + V_{in}/R_{off}$	2	1	0
$3 * V_{in}/R_{on}$	3	1	1

Table 3.2: Addition of two unsigned binary numbers $A = 11$ and $B = 01$

Time	Input Bit			$l = \lfloor I_{out}/I_{ref} \rfloor$	Output logic	
t	a_t	b_t	c_{t-1}	where $I_{ref} = V_{in}/R_{on}$	Carry ($Q_1 = c_t$)	Sum ($Q_0 = S_t$)
0	1	1	0	2	1	0
1	1	0	1	2	1	0

3.3 Designing Adders Based on Current-Comparison

We augment a memristor crossbar with associated CMOS peripheral circuits to implement scalable arithmetic as well as logical computing units where the crossbar layer is employed as a processing and storing unit, and the peripheral circuits act as a control unit for writing

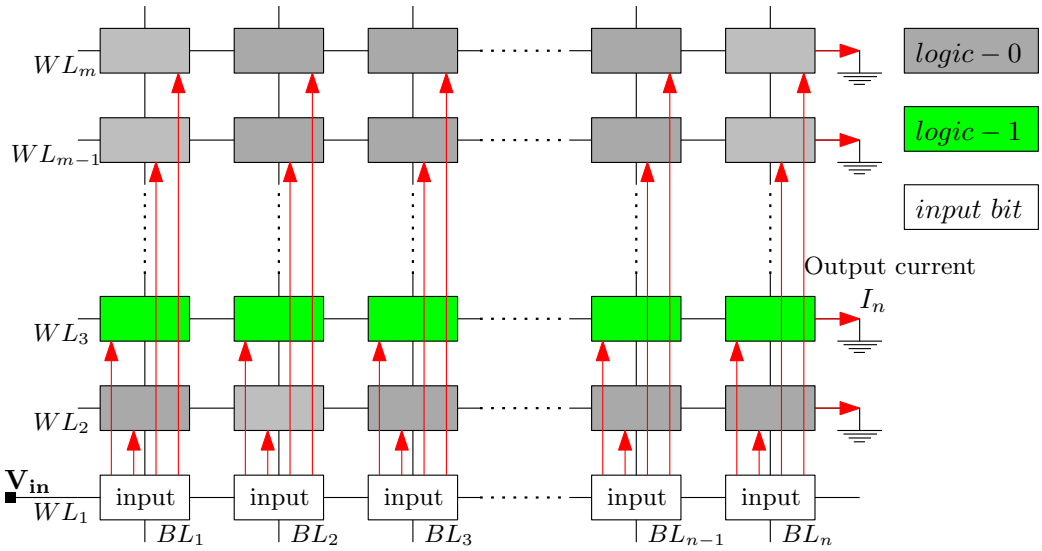


Figure 3.2: Direction of current flow in the crossbar from WL_1 to OCU

input bits into the crossbar and generation of output bits by processing the output current.

In order to execute any operation in an $M \times N$ crossbar, at any cycle k input bits are fed in terms of low/high memresistance to k memristors connected to the bottom-most word-lines (WL). The rest of $N-k$ memristors, connected to it are set to M_{off} state. The remaining $M(N-1)$ memristors are set to either in M_{off} or M_{on} state, depending on the applications. One peripheral circuit is connected to each of the $(M-1)$ WLs to generate the output bits. When a fixed input voltage V_{in} is applied to the bottom-most WL, the corresponding output bits are generated for that cycle. The current flow in a crossbar configuration is depicted in Figure 3.2, demonstrating the direction of current originating from WL_1 (the input line) to all other $m-1$ WLs in an $m \times n$ crossbar array. During each cycle, a single WL is selectively activated by setting all the connected memristors to a logical state of 1, thereby generating an output. Consequently, the output current is contributed by a maximum of n conducting paths.

The computation is done in analog domain to achieve reduced computation time and improved energy efficiency. Rather than applying different voltages to WLs and BLs as suggested in (HCJ18; EFR15) at different time instants, the proposed design applies a constant voltage to a fixed WL. In order to generate the output values, we employ a simple peripheral circuit to generate the output logic bits by analyzing the output current of the

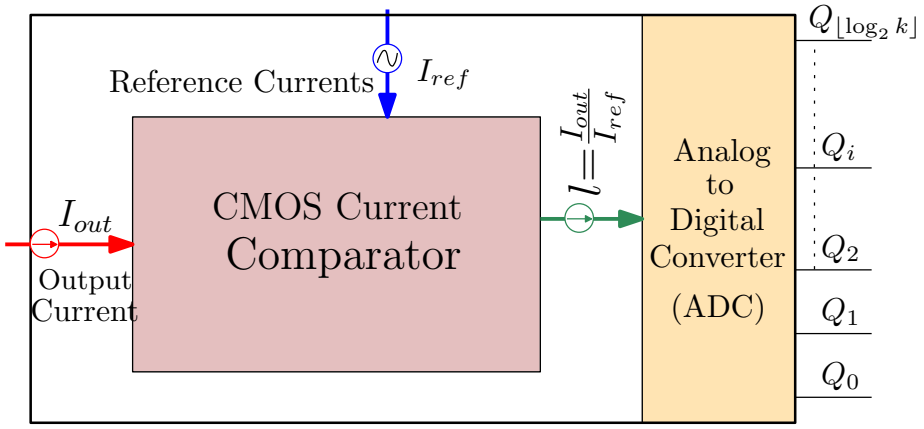


Figure 3.3: Block diagram of an *Output Computing Unit (OCU)*

crossbar, as described below.

3.3.1 Design of output computing unit (OCU)

Here, we propose a simple peripheral circuit connected to each BL to process the output current for generating the output logic, and is termed henceforth as the output computing unit (OCU). The underlying working principle of OCU is to count the number of high current conducting paths and to convert it into digital bits. In an $M \times N$ crossbar, at any cycle the output current is sensed in any of the $N-1$ BLs and the OCU connected to that BL generates the output bit for that cycle. An OCU may comprise an analog multi-valued current (voltage) comparator followed by a CMOS analog to digital converter (ADC) as shown in Figure 3.3. A multi channel current input ADC can also be employed as an OCU.

In the proposed method, $I_{high} = \frac{V_{in}}{2M_{on}}$ and $I_{low} = \frac{V_{in}}{M_{off}+M_{on}}$ denote the current passing through ON- and OFF-memristance paths, V_{in}, M_{on} and M_{off} denote the input voltage, ON and OFF memristance, respectively. At any cycle, if l inputs are 1, then the total output current at BL is $I_{tot} = l * I_{on} + (N-l) * I_{off}$. An analog current comparator can compute l by comparing $\frac{I_{tot}}{I_{on}}$ as $I_{on} >> I_{off}$. Next, l is converted into an equivalent (rounded) binary number by an ADC. Table 3.3 present a notation-list used in this chapter.

Addition is one of the most elementary and frequently used operations in almost all applications. Therefore, efficient and fast in-memory addition technique in CIM is very essential for memristor-based processors. Depending on the application, numerous binary adders have been proposed such as ripple carry adder (RCA), carry-lookahead adder (CLA)

Table 3.3: Glossary of notations

Description	Notation
$\lceil \log_2(k + \log_2 k) \rceil$	α_k
OCU having k output pins	OCU^k
i^{th} output bit of OCU connected to BL_{j+1}	Q_i^j
Input voltage to WL_1	V_{in}

and carry-save adder (*CSA*) (Man79). Here, we propose the design of a fast and scalable adder using hybrid-memristive crossbar. Two types of adder circuits are presented that significantly reduce the number of execution cycles as well as crossbar area compared to prior art.

3.3.2 Fast multi-operand addition inspired by Carry-Save Adder

A scalable design of a fast adder is proposed where k n-bit integers are added in parallel. Thus, it is a multi-operand adder where in each step, k bits are added starting from the least significant bit (LSB) and moving towards the most significant bit (MSB), with diagonal carry propagation as in a CSA.

In order to execute the addition of k n-bit numbers, we require a crossbar of size $2 \times (k + \alpha_k)$ where $\alpha_k = \lceil \log_2(k + \log_2 k) \rceil$. At any cycle t, the t^{th} bits of k inputs and α_k carry bits are written into the memristors connected to WL_1. The remaining memristors, connected to the other word line WL_2, are set to *logic*-1. When an input readout voltage V_{in} is applied to WL_1, the output bits for the t^{th} cycle are generated by the OCU. The LSB of the OCU represents the t^{th} sum bit, and the rest of the output bits represent the carry bits are to be added with starting from $(t + 1)^{th}$ bit and aligning them towards the MSB position, and summing them in the subsequent execution cycles. The complete procedure is illustrated with the next two examples.

Example 3.3.1. *Consider the addition of two unsigned binary numbers* $A = 1101$ *and* $B = 1001$ *on a* 2×3 *crossbar connected to an OCU having two output pins. In the first cycle, the LSB, i.e.,* 0^{th} *bit of both the numbers are added and the generated carry bit is shifted to the left, i.e., to the* 1^{st} *bit position. In the next cycle, all bits in the* 1^{st} *bit position*

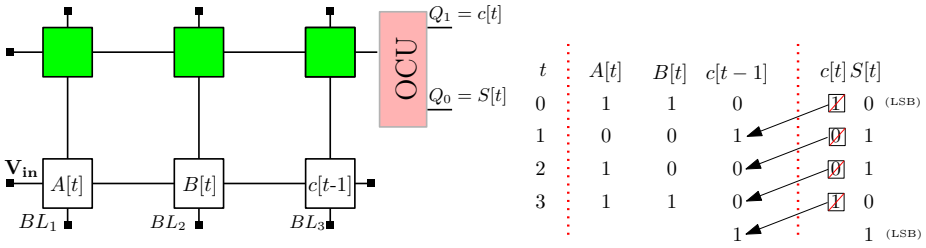


Figure 3.4: Feeding binary inputs in the t^{th} cycle for adding two unsigned binary 4-bit numbers, and the corresponding five execution cycles

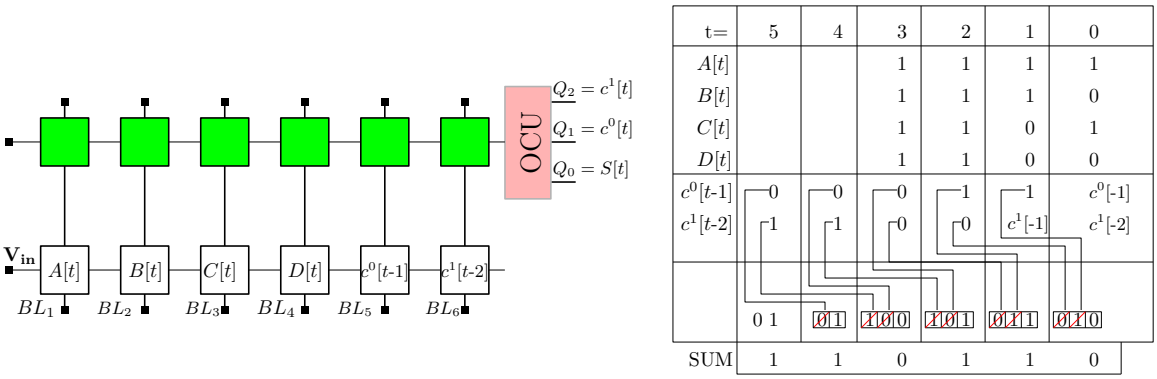
Table 3.4: Input mapping on the crossbar and the output generation at each time step of addition

t	Memristor cells				Output at OCU	
	Memristors connected to WL_1			Memristors connected to WL_2		
	$M_{11} \leftarrow A[t]$	$M_{12} \leftarrow B[t]$	$M_{13} \leftarrow c[t-1]$	M_{21}, M_{22}, M_{23}	$Q_1 = c[t-1]$	$Q_0 = S[t]$
0	1	1	0* (let)	1*	1	0
1	0	0	1	1*	0	1
2	1	0	0	1*	0	1
3	1	1	0	1*	1	0
4	0*	0*	1	1*		1

The entries marked with * are predetermined logic bits used for execution.

along with the previously generated carry bits are added together. This process continues for five cycles to generate all the sum bits, i.e., four cycles for adding the four bits of the input numbers and one extra cycle for the last shifted carry bit.

In order to implement this addition on a memristive crossbar, we set, at $t = 0$, $A[0] = 1$ *and* $B[0] = 1$ *into memristor cells* M_{11} *and* M_{12}*, respectively. The initial carry bit* $c[-1] = 0$ *is written into memristor cell* M_{13}*. Logic-*1 *is written in* M_{21}*,* M_{22} *and* M_{23}*. When* V_{in} *is applied to* WL_1*,* $Q = 10$ *is generated at OCU where* $Q_0 = S[0] = 0$ *is the sum bit and* $Q_1 = c[0] = 1$ *is the carry bit for the next cycle. In the consecutive cycle at* $t = 1$*,* $A[1] = 0$*,* $B[1] = 0$ *and* $c[0] = 1$ *are written into memristor cells* M_{11}*,* M_{12} *and* M_{13}*, respectively. Logic-*1 *is written to the cells connected to* WL_2 *and addition is performed. Thus the process is completed in five execution cycles as shown in Figure 3.4 and the output bits after each execution cycle is shown in Table 3.4.*



t=	5	4	3	2	1	0
A[t]			1	1	1	1
B[t]			1	1	1	0
C[t]			1	1	0	1
D[t]			1	1	0	0
c^0[t-1]	0	0	0	1	1	c^0[-1]
c^1[t-2]	1	1	0	0	c^1[-1]	c^1[-2]
	0 1	∅ 1	1 0 0	1 0 1	0 1 1	0 1 0
SUM	1	1	0	1	1	0

Figure 3.5: Crossbar mapping and the execution cycles for adding four unsigned binary numbers

Example 3.3.2. *Let us consider another example where four binary numbers* $A = 1111$, $B = 1110$, $C = 1101$ *and* $D = 1100$ *are added together. The addition can be performed on a* 2×6 *crossbar with an OCU having three output pins. At* t^{th} *cycle, the four* t^{th} *input bits along with two* $(\log_2 4)$ *carry bits, generated at* $(t-1)^{th}$ *and* $(t-2)^{th}$ *cycle, are added together. Three output bits are generated including the* t^{th} *output bit, and two are carry bits that are added at* $(t+1)^{th}$ *and* $(t+2)$ *cycle, as shown in Figure 3.5. The memristor states and the execution cycles are shown in Table 3.5.*

Algorithm 1 captures the detailed execution of unsigned addition inspired by CSA, for adding k n-bits numbers. The whole addition takes $n + \alpha_k$ cycles where n cycles are required for performing additions in n bit-positions and α_k cycles for adding the shifted carry bits (the longest carry propagation being α_k).

The design is scalable in the sense that addition with large values of k and n can be performed on a memristive array of a given size, by adopting a divide-and-conquer strategy that relies on judicious partitioning of the input space. However, for a particular application, the maximum value of k and n that can be handled in a single pass depend on the available crossbar size as well as the number of output pins of the OCU (resolution of the ADC). The cost of the ADC increases sharply with increasing converter resolution. Therefore, to fit the problem into the available crossbar size and ADC resolution, k and n can be partitioned into smaller sets and addition can be done separately. For example, if $k = 60$ and ADC resolution is 4-bit, then k can be divided into four groups, each containing $2^4 - 1 = 15$ binary numbers. Firstly, these four groups are added separately and the outputs are added in the final step. Therefore, the design is referred to as *dynamic design* and in order to

Table 3.5: State of the memristors at each cycle while adding four 4-bit binary numbers.

t	Memristor cells							Output at OCU		
	Cell connected to WL_1						Cell(s) connected to WL_2			
	M_{11} ↑ $A[t]$	M_{12} ↑ $B[t]$	M_{13} ↑ $C[t]$	M_{14} ↑ $D[t]$	M_{15} ↑ $c^1[t-2]$	M_{16} ↑ $c^0[t-1]$	$M_{21} \dots M_{26}$	$Q_2 = c^1[t]$	$Q_1 = c^0[t]$	$Q_0 = S[t]$
0	1	0	1	0	0*	0*	1*	0	1	0
1	1	1	0	0	0*	1	1*	0	1	1
2	1	1	1	1	0	1	1*	1	0	1
3	1	1	1	1	0	0	1*	1	0	0
4	0*	0*	0*	0*	1	0	1*		0	1
5	0*	0*	0*	0*	1	0	1*		0	1
5	0*	0*	0*	0*	1	0*	1*			1

Algorithm 1: Pseudo-code of fast multi operand binary addition of k n-bit unsigned integers on a memristive crossbar, inspired by CSA.

Data: k n-bit unsigned binary numbers, a crossbar of size $2 \times (k + \alpha_k)$, V_{in}, OCU^{α_k};
Result: Final sum bits $S[n]$, $0 \leq n \leq (n + \alpha_k)$

for *time* $t = 0$ *to* $n - 1$ **do**
    Write t^{th}-bit of all k numbers as memristor state in
    $M_{1,1}$ to $M_{1,k}$;
    Write α_k carry bits, each generated from last α_k cycles into memristors
    $M_{1,k+1}$ to $M_{1,k+\alpha_k}$
    for $m = 1$ *to* α_k **do**
        Carry bit generated at pin Q_m in t^{t-m}th cycle is written to memristor $M_{1,k+m}$;
    Write *logic*-1 to memristors $M_{2,1}$ to $M_{2,k+\alpha_k}$;
    Apply voltage V_{in} to WL_1;
    Final t^{th} sum bit is generated at $Q_0 = S[t]$;
    Bit generated on Q_1 to $Q_{\alpha_k - 1}$ are the carry bits to be used in the next α_k cycles;
for $t = n$ *to* $n + \alpha_k$ **do**
    The carry bits to be executed in t^{th} cycles are added to produce the t^{th} sum bit and carry bits for future execution;

Algorithm 2: Crossbar mapping for fast multi operand binary addition of k n-bit unsigned integers on a memristive crossbar, inspired by CSA.

Data: k, n-bit binary numbers,
$A^{k-1} = a^{k-1}[n-1]a^{k-1}[n-2]\dots a^{k-1}[1]a^{k-1}[0]$;
$A^{k-2} = a^{k-2}[n-1]a^{k-2}[n-2]\dots a^{k-2}[1]a^{k-2}[0]$;
$\vdots$
$A^{1} = a^{1}[n-1]a^{1}[n-2]\dots a^{1}[1]a^{1}[0]$;
$A^{0} = a^{0}[n-1]a^{0}[n-2]\dots a^{0}[1]a^{0}[0]$;
A crossbar of size $2 \times (k + \alpha_k)$, where $\alpha_k = \lfloor \log_2(k + \log_2 k) \rfloor$;
Voltage source V_{in};
$OCU^{\alpha_k} \longrightarrow$ OCU having α_k output pins;
Result: Final sum bits $S = s[n+\alpha_k]s[n+\alpha_k-1]\dots s[1]s[0]$;

```
for time, t = 0 to n − 1 do
    for j=0 to k − 1 do
        M_{1,j+1} ← a^j[t]; % mapping of t^th bit of j^th input integer
    for m = 1 to α_k do
        M_{1,k+m} ← c^{m−1}[t − m]; % mapping of previously generated carry bits
    for l = 1 to (k + α_k) do
        M_{2,l} ← logic-1;
    WL_1 ← V_in;
    s[t] ← Q^0[t]; % generation of t^th sum bit
    for p = 0 to α_k − 1 do
        c^p[t] ← Q^{p+1}[t]; % carry bits for next state operations
for t = n to n + α_k do
    for i = 1 to t do
        M_{1,i} ← logic-0;
    for j = 1 to α_k do
        M_{1,t+j} ← c^{j−1}[t − j];
    WL_1 ← V_in;
    S[t] ← Q^0[t]; % generation of t^th sum bit
    for p = 0 to α_k − 1 do
        c^p[t] ← Q^{p+1}[t]; % carry bits for next state operations
```

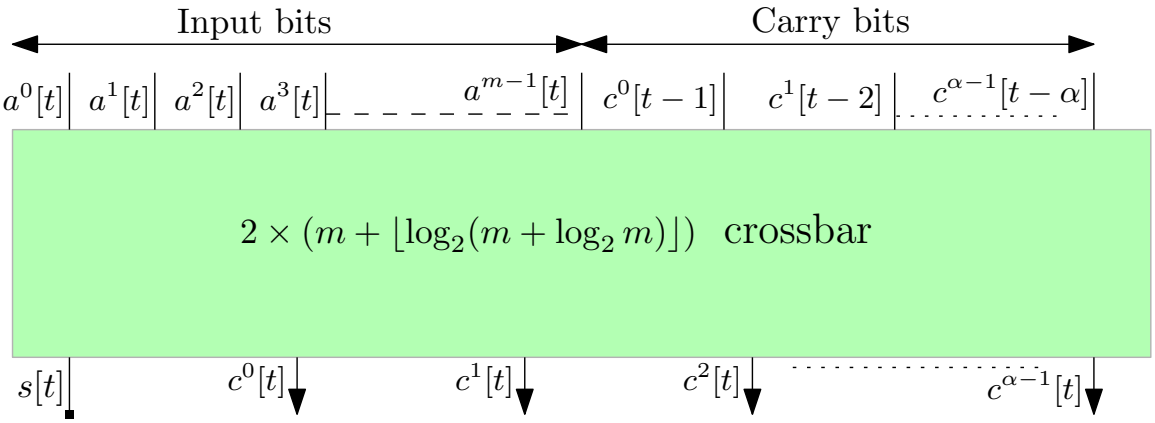


Figure 3.6: Block diagram of a multi-operand adder inspired by CSA on a $2 \times (k + \alpha_k)$ crossbar.

execute the addition of k unsigned binary numbers in one pass, the OCU ought to have α_k output pins and crossbar size greater than or equal to $2 \times (k + \alpha_k)$. Addition of operands with large word size (n) can be handled on a crossbar of smaller size by increasing the number of execution cycles.

The block diagram of the proposed CSA-based unsigned adder circuit on a memristive crossbar is shown in Figure 3.6. Table 3.6 shows the size of the maximum carry bits, the required crossbar size, the size of OCU and the number of cycles required for complete execution against different input binary numbers (referred to as model size). The most notable point is that the number of execution cycles primarily depends on the size of the input numbers (n) rather than count of the input numbers (k). Therefore, the proposed circuit is suitable for multi-operand addition.

Table 3.6: Characteristics of the proposed CSA-based adder circuit

Module size k	# carry bits $\alpha_k - 1$	Crossbar size $2 \times (k + \alpha_k)$	#OCU pins α_k	#Cycles $n + \alpha_k - 1$
2	1	2×3	2	n+1
3	2	2×5	3	n+2
4	2	2×6	3	n+2
5	2	2×7	3	n+2
6	3	2×9	4	n+3
7	3	2×10	4	n+3
8	3	2×11	4	n+3
16	4	2×20	5	n+4
32	5	2×37	6	n+5
64	6	2×70	7	n+6
128	7	2×135	8	n+7
256	8	2×264	9	n+8

3.3.3 Memristive Addition inspired by Carry-Look Ahead Adder

The design of a general-purpose CLA suffers from high hardware overhead because of the fact that all carry bits need to be generated directly from the inputs. However, in memristive realization, the OCU unit can be conveniently used for this purpose. When two input bits a and b are applied to the memristive crossbar as proposed in Section 3.3.2, the two output pins of the OCU from the LSB-side directly provide the carry-propagate P ($P = a \oplus b$) and the carry-generate G ($G = a \wedge b$) signal. Leveraging this property, a CLA-inspired adder is proposed where the $P[t]$s and $G[t]$s are produced for each bit position of the binary inputs. Next, we generate carry bits for each bit position using previously generated $P[t]$s and $G[t]$s. In the third cycle, the final sum is obtained by using the generated carry bits.

Cycle-1: Carry-propagate ($P[t]$) and Carry-generate ($G[t]$) computation

Algorithm 3 explains the generation of $P[t]$ and $G[t]$ for an n-bit CLA-inspired adder while adding two unsigned binary numbers $A[n]$ and $B[n]$ by employing a crossbar of size $(n + 1) \times 2n$ with n OCUs in one cycle.

Algorithm 3: Pseudo-code for generating Carry-propagate ($P[t]$) and Carry-generate ($G[t]$) bits for CLA-inspired addition of two n-bit numbers

Data: Two n-bit unsigned numbers;
A crossbar of size $(n+1) \times 2n$, OCU^2;
Result: $P[t]$ and $G[t]$, $0 \leq t \leq n-1$;

Make bit-wise addition of all the bits simultaneously such that OCU connected to BL_{t+1} produces the addition for t^{th} bit positions where Q_0 (LSB) produces carry propagate and Q_1, carry generate;
$P[t] = Q_0^{t+1}$;
$G[t] = Q_1^{t+1}$;

Algorithm 4: Crossbar mapping for generating Carry-propagate ($P[t]$) and Carry-generate ($G[t]$) bits for CLA inspired addition of two n-bit numbers

Data: Two n-bit binary numbers, $A = a_{n-1}...a_2a_1a_0$ and $B = b_{n-1}...b_2b_1b_0$;
A crossbar of size $(n+1) \times 2n$;
n units OCU;
Result: $P[t]$ and $G[t]$, $0 \leq t \leq n-1$;

for t=0 *to* $(n-1)$ **do**
    $M_{1,2t+1} \leftarrow A[t]$;
    $M_{1,2t+2} \leftarrow B[t]$;
    for j= 1 *to* 2 **do**
        $M_{(2+t),(2t+j)} \leftarrow$ *logic*-1;
logic-0 is written to the rest of the memristors in the crossbar;
$WL_1 \leftarrow V_{in}$;
for t=0 *to* $n-1$ **do**
    $P[t] = Q_0^{t+1}$;
    $G[t] = Q_1^{t+1}$;

Cycle-2: Carry $c[t]$ generation

In the second cycle, the carry bits for each bit position of the two numbers are determined ahead of time utilizing $P[t]$s and $G[t]$s. In Algorithm 6, we present the generation of carry bits $c[t],\ 1 \leq t \leq n$ using $P[t]$s and $G[t]$s. A crossbar of size $(n+1) \times 2n$ is required to generate n carry bits.

Algorithm 5: Pseudo-code for the generation of Carry bits $c[t]$ using $P[t]$s and $G[t]$s generated at Cycle-1.

Data: $P[t]$ and $G[t]$, $0 \leq t \leq n-1$ produced in Algorithm 3;
A crossbar of size $(n+1) \times 2n$;
Result: List $c[t]$ $1 \leq t \leq n+1$;

1 Implementation of carry function $c[1]$ is effected on the memristive-crossbar by setting two parallel paths $G[0]$ and $c[0]P[0]$ from input line WL_1 to output line BL_1 using two rows and two columns;
2 To synthesize $c[2]$, the output line of $c[1]$ is utilized to set two parallel path $G[1]$ and $c[1]P[1]$ from input line WL_1 to output line BL_2 using one extra row and one extra column;
3 In order to generate each of the higher-order carry bits, one additional row and column is required;
4 This process is repeated through n steps;

Cycle-3: Final addition and generation of sum bits $S[t]$

In the third cycle, the final sum bits are generated by adding two inputs and carry bits generated in Cycle-2. Only the LSB bit of each OCU is considered as corresponding sum bit. However, for the leftmost bit position, the MSB bit of the OCU is treated as the final sum bit. Algorithm 8, presents the generation of final sum-bits $S[t],\ 0 \leq t \leq n$ using the generated carry bits. A crossbar of size $(n+1) \times 3n$ is required to generate the final sum with n OCU units.

Example 3.3.3. *Here, we perform the addition of two unsigned binary numbers $A = 1110$ and $B = 1001$. Figure 3.7 shows the input mapping as well as the generation of $P[t]$ and the $G[t]$ for each bit position in the first cycle. Carry bits are generated in the second cycle and four carry generation functions are given below.*

1. $c[1] = G[0] + c[0]P[0]$
2. $c[2] = G[1] + G[0]P[1] + c[0]P[0]P[1]$
3. $c[3] = G[2] + G[1]P[2] + G[0]P[1]P[2] + c[0]P[0]P[1]P[2]$
4. $c[4] = G[3] + G[2]P[3] + G[1]P[2]P[3] + G[0]P[1]P[2]P[3] + c[0]P[0]P[1]P[2]P[3]$

Algorithm 6: Crossbar mapping for the generation of Carry bits $c[t]$ using $P[t]$s and $G[t]$s generated at cycle-1.

Data: $P[t]$ and $G[t]$, $0 \leq t \leq n-1$ produced in Algorithm 3;
A crossbar of size $(\binom{n+1}{2} - n + 1) \times (\binom{n+1}{2} - 1)$;
Result: List $c[t]$ $1 \leq t \leq n-1$;

$count_O = 0$;
for $t = (n-1)$ *to* 1 **do**
  $k = \frac{n(n+1)}{2} - \frac{(t+1)(t+2)}{2}$;
  $a = \lfloor \frac{t+1}{2} \rfloor + 1$
  $b = \lceil \frac{t+1}{2} \rceil$
  $count_I = 1$;
  x, y=0;
  for $j \leftarrow (n-1-count_O)$ *to* 1 **do**
    if $j = n-1-count_O$ **then**
      $M_{1,(k+1)} \leftarrow P[j-1]$
      $M_{1,(k+j+1)} \leftarrow G[j-1]$
      $count_I$=$count_I + 1$;
    else if *($count_I \geq 2$) & ($count_I$=even)* **then**
      δ=$count_I/2$;
      $x = (k+1+\delta-count_O)$;
      $y = (k+\delta)$;
      $M_{x,y} \leftarrow P[j-1]$
      $M_{x+j,y} \leftarrow G[j-1]$
      $count_I$=$count_I + 1$;
    else
      $M_{x,y+1} \leftarrow P[j-1]$
      $M_{x,y+j+1} \leftarrow P[j-1]$
      $count_I$=$count_I + 1$;
    $M_{(k+a-count_O),(k+b)} \leftarrow c[0]$;
    $WL_{k+t+1-count_O} \leftarrow c[t]$;
  $count_O = count_O + 1$;

Algorithm 7: Pseudo-code for the generation of Sum bits $S[t]$, $0 \leq t \leq n$

Data: n carry bits $c[t]$, $1 \leq t \leq n+1$ produced in Algorithm 6;
Initial carry $c[0]$, $(n+1) \times 3n$ crossbar, OCU^2;
Result: Final $n+1$ sum-bits.

1 The t^{th} bit of two input numbers, and the carry bit $c[t]$ are written at memristors $M_{1,2t+1}$, $M_{1,2t+2}$, and $M_{1,3(t+1)}$, respectively;
2 Addition of these three bits are executed using BL_{t+1};
3 The final t^{th} sum bit is generated at Q_0^{t+1};
4 Additions for all bit-positions are executed simultaneously;

Algorithm 8: Crossbar mapping for the generation of Sum bits $S[t]$, $0 \leq t \leq n$

Data: n-2 carry bits $c[t]$, $1 \leq t \leq n-1$ produced in Algorithm 6;
Initial carry $c[0]$;
A crossbar of size $(n+1) \times 3n$;
n units of OCU;
Result: Final $n+1$ sum bits.

1 **for** $t = 0$ *to* $n-1$ **do**
2 $M_{1,2t+1} \leftarrow A[t]$;
3 $M_{1,2t+2} \leftarrow B[t]$;
4 $M_{1,3(t+1)} \leftarrow c[t]$;
5 **for** $j = 1$ *to* 3 **do**
6 $M_{(2+t),(3t+j)} \leftarrow$ *logic*-1;
7 *logic*-0 is written to the rest of the memristors in the crossbar;
8 $WL_1 \leftarrow V_{in}$;
9 **for** $j = 0$ *to* $n-1$ **do**
10 $S[j] = Q_0^{j+1}$;
11 $S[n] = Q_1^n$;

Table 3.7: Characteristics of a CLA-inspired adder

Word size	Crossbar size			#OCU used
	Cycle-1	Cycle-2	Cycle-3	
n	$(n+1) \times 2n$	$(n+1) \times 2n$	$(n+1) \times 3n$	n
4	5×8	5×8	5×12	4
6	7×12	7×12	7×18	6
8	9×16	9×16	9×24	8
16	17×32	17×32	17×48	16
32	33×64	33×64	33×96	32
64	65×128	65×128	65×192	64

The emulation of carry functions for $c[1]$, $c[2]$, $c[3]$ *and* $c[4]$ *require an* 5×8 *crossbar and the mapping is shown in Figure 3.8. The final output is computed in the third cycle and the input mapping to the crossbar and final sum generation are shown in Figure 3.9.*

The block diagram of an n-bit CLA-inspired fast adder circuit is shown in Figure 3.10. The addition is completed in three cycles which need crossbars of different sizes. However,in practice, all three cycles can be executed on a single crossbar (the one with the largest size) one after another, thus limiting the hardware overhead. Table 3.7 shows the size of the maximum carry bits, required crossbar size, the number of OCU units used, and the number of the cycles needed for complete execution against different input number size. Compared to the CSA-based adder described earlier which requires $\mathcal{O}(n)$ cycles, the CLA-based approach completes addition in only three cycles. However, it needs a larger memristive crossbar for emulation as well as a complex peripheral control circuit. Furthermore, it is not suitable for handling addition of more than two operands, concurrently.

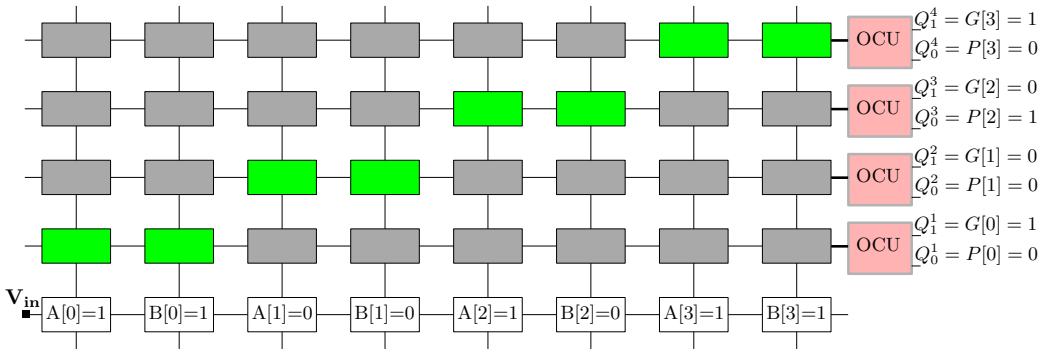


Figure 3.7: Mapping on a 5×8 crossbar with four OCU units for generating carry-propagate ($P[t]$) and carry-generate ($G[t]$) while adding two 4-bit numbers

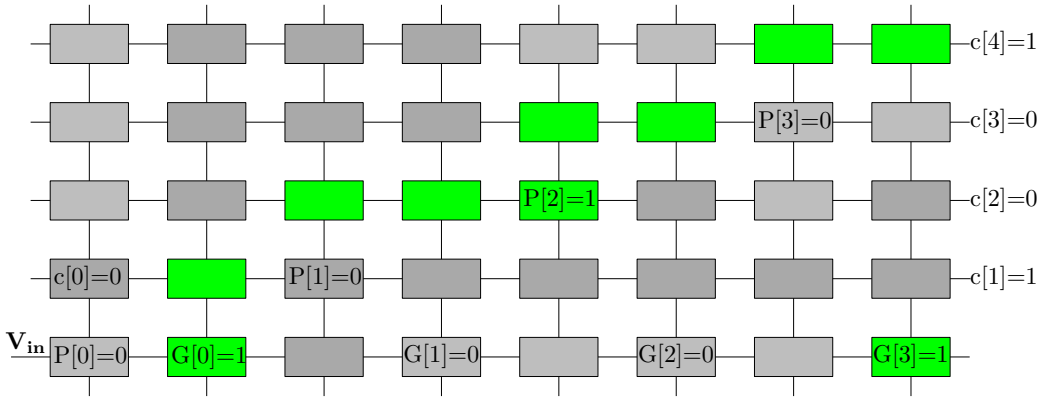


Figure 3.8: Generation of carry bits $c[1]$, $c[2]$, $c[3]$ and $c[4]$ on a 5×8 crossbar

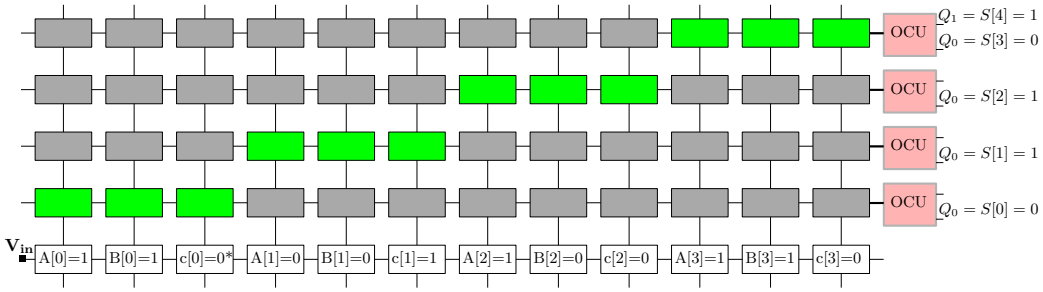


Figure 3.9: Generation of the final sum bits $S = 10110$ on a 5×12 crossbar

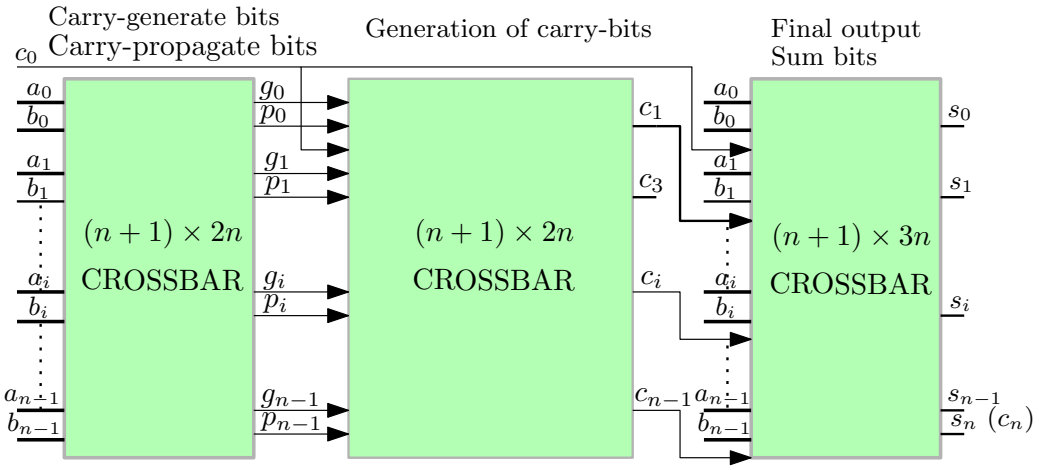


Figure 3.10: Block diagram of a CLA-inspired n-bit adder

3.4 Evaluation of our Memristive Adders

3.4.1 Simulation results

Table 3.8: Comparative results with prior works

Operation	(TGY+17),(YT18)		**Proposed method**		
	#mem	cycle	crossbar size	#mem	cycle
Full Adder	15M	13	2×3	8M	2
4-bit RCA	57M	49	2×3	29M	8
4-bit CLA	109M	101	7×12	50M	6
n-bit RCA	$14n$+1	$12n$+1	2×3	$7n$+1	$2n$
n-bit CLA	N.A.	N.A.	$(n+1) \times 3n$	$7(n^2+1)$	6

We define the step current I_Δ as $v/2R$, the amount of current passing through an active path. All other possible values of current are multiples of I_Δ. Now, I_Δ depends only on V_{in}, the input voltage and R_{ON} of a memristor. A modified current comparator can be configured on the basis of these two parameters so that the step current is large enough.

Example 3.4.1. *The spice code for simulating the five parallel paths in a 5 × 5 crossbar is presented here.*

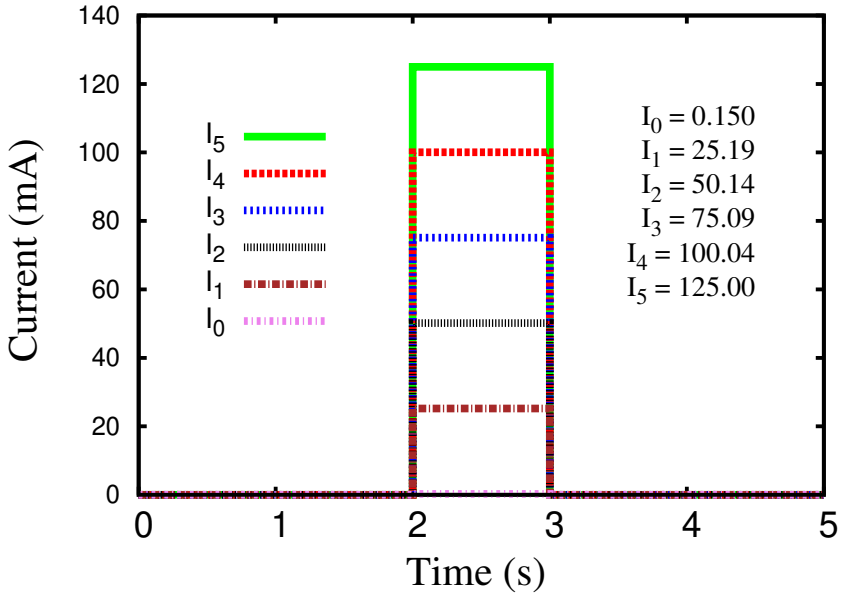


Figure 3.11: Simulation results with five discrete current values.

```
* HP MEMRISTOR MODEL USING JOGLEKAR WINDOW FUNCTION
 * Connections:
 *TE: Top Electrode
 *BE: Bottom electrode
 *XSV: External connection to plot state variable that is not used otherwise

 .subckt Mem_joglekar TE BE XSV

 * Ron: minimum device resistance
 *Roff : maximum device resistance
 *D : Width of the thin film
 *uv: Dopant mobility
 *p: parameter for window function
 *x0: state variable initial value

 .param Ron=100 Roff=100k x0=0.56 D=16n p=7 uv=40F
```

```
 *JOGLEKAR window function definition
 .func f(V1)=1-pow((2*V1-1),(2*p))

 *MEMRISTOR I-V relationship
 .func IVRel(V1,V2)=V1/(Ron*V2+Roff*(1-V2))

 *circuit to determine state variable, Gx Linear voltage controlled current source
 Gx 0 XSV value={I(Gmem)*Ron*uv*f(V(XSV,0))/pow(D,2)}
 Cx XSV 0 {1}
 .ic V(XSV)=x0

 *Current sourcerepresenting memristor
 Gmem TE BE value={IVRel(V(TE,BE),V(XSV,0))}
 .ends Mem_joglekar

Xmemristor1 p1 q1 XSV Mem_joglekar x0=1
Xmemristor2 p1 q2 XSV Mem_joglekar x0=1
Xmemristor3 p1 q3 XSV Mem_joglekar x0=1
Xmemristor1 p1 q4 XSV Mem_joglekar x0=1
Xmemristor1 p1 q5 XSV Mem_joglekar x0=1

Xmemristor4 q1 p2 XSV Mem_joglekar x0=1
Xmemristor5 q2 p2 XSV Mem_joglekar x0=1
Xmemristor6 q3 p2 XSV Mem_joglekar x0=1
Xmemristor4 q4 p2 XSV Mem_joglekar x0=1
Xmemristor5 q5 p2 XSV Mem_joglekar x0=1

vin p1 0 5
R1 p2 0 100
.tran 0.1
.end
```

We perform resistive spice simulation using LTSpice with $V = 5V$, $R_{ON} = 100\Omega$ and $R_{OFF} = 100k\Omega$. We measure the sum of current-values passing through five parallel

paths in a 5×5 crossbar, as shown in Fig. 3.11, where $I_\Delta = 25mA$. The four other values of current are consecutive integer multiples of I_Δ. As $R_{OFF} >> R_{ON}$, the current passing through the off memristor (I_{OFF}) is negligible in comparison to I_Δ, so the effect of I_{OFF} is not considered while calculating I_Δ. Further, we observed that the value of I_Δ is nearly constant when $\frac{R_{ON}}{R_{OFF}}$ is very small ($< 10^{-3}$), but becomes unstable as $\frac{R_{ON}}{R_{OFF}}$ increases significantly.

3.4.2 Discussion

In Table 3.8, we compare the results of our method with that in (TGY+17) where both the methods include the time for writing input-bits as well as the execution. For the memristor count, we have included those in the pre-defined output block where the final output of the module is to be stored. Our method reduces not only the memristor count, but also the number of time cycles significantly.

For implementing in-memory module, available crossbar size is very important. To the best of our knowledge, no previous attempt was made to scale the module size based on available memory. In our design, the size of the module can be optimized depending on the task schedule and available memory size. For the *CLA*, we generate only the rightmost n carry bit in Step II, and then c_n only in the final step to optimize the crossbar size.

3.5 Concluding Remarks

We have proposed here an area-aware binary adder suitable for in-memory computation on a hybrid-memristor crossbar, where the output logic is determined based on differential output currents. A simple CMOS-based *OCU* comprising of a current comparator and an *ADC* is capable of analyzing the output current to determine the output. The proposed method is fast, and the size of a module can be tuned according to available memory space. Addition is one of the key operations for updating node-weight in neuromorphic computation for which CIM would be promising candidate.

CHAPTER

FOUR

MEMRISTIVE CROSSBAR ARCHITECTURES FOR SUBTRACTOR, MULTIPLIER AND LOGICAL MODULES

Contents

4.1 Introduction

A 2D memristive crossbar can be used to store multi-valued memory states corresponding to the analog variation of current-induced resistance through memristor cells. The integration of CMOS components with memristors further widens their design space for managing various complex systems, and such hybridization supports a potential platform for enabling computation-in-memory (CIM).

In this chapter, we present a scalable implementation of *in-memory* arithmetic logic units on a memristive crossbar. Digital designs being inherently modular, our objective here is to build basic circuit modules and then cascade them to realize a large-size complex unit. In particular, we propose designs for arithmetic and logical circuits on a memristor-based hybrid crossbar network that relies on current sensing along with some analog peripheral circuits. Thus, computation is performed in a mixed domain, i.e., a binary input is mapped as a state of a memristor in the crossbar topology, whereas the outcome of the arithmetic and logic operations appear in the analog domain, a hybridized strategy that leads to reduced computation time and improved energy efficiency. The peripheral circuits capture output currents and produce either intermediate logic outputs to be mapped again to the crossbar for the next cycle of execution or to provide the final digital outputs. The proposed designs are scalable and programmable in tune with the specification and available crossbar area. This technique allows parallel computation on large-size binary operands and is thus suitable for machine learning applications. The major contributions of our work on designing memristive circuits employing CIM for various operations are listed below:

- shift-and-add multiplication of two unsigned n-bit binary numbers, in $\mathcal{O}(n)$ cycles. The multiplication procedure takes $(2n+\lceil\log_2(n+\log_2 n)\rceil)$ cycles, employing a crossbar of size $(2n+\lfloor\log_2(n+\log_2 n)\rfloor)\times(n+\lfloor\log_2(n+\log_2 n)\rfloor)$;
- subtraction involving two unsigned n-bit binary numbers. It requires three cycles involving a crossbar of size $(n+1)\times 2n$, $(n+1)\times 5n$ and $(n+1)\times 2n$, respectively;
- logical operations on n-bit words. NOT and AND operations are executed in a single cycle whereas an OR operation would need four cycles.

4.2 Binary Subtraction and Multiplication

Chapter 3 addresses the design of a fast and scalable adder using hybrid-memristive crossbar as addition is one of the most elementary and frequently used operations in almost all

applications. Therefore, efficient and fast in-memory addition techniques in CIM are very essential for memristor-based processors. Depending on the application, we have designed different types of memristive crossbar based binary adders such as ripple carry adder (RCA), carry-lookahead adder (CLA) and carry-save adder (CSA) (Man79). In this chapter, we next present the design of two more arithmetic operations, namely binary subtraction and multiplication of two unsigned binary numbers using the CSA-inspired adder module of Chapter 3.

4.2.1 Binary subtractor

The design of a subtractor with $2's$ complement for unsigned binary subtraction is proposed in a memristive crossbar. The subtraction is completed in three cycles as follows.

1. $1's$ complement of the subtrahend is generated.
2. The minuend, the computed $1's$ complement of the subtrahend and constant 1 are added.
3. If the final carry bit is 1,
   - then the carry bit is discarded and the rest of the bits are saved as result (which is a positive number).
4. else, the resultant bits output the $2's$ complement of the final answer (this may be considered as a negative number).

Generation of 1′s complement, 2′s complement, and addition are thus necessary for implementing subtraction of two unsigned binary numbers. The memristive circuit proposed in Section 4.2 can be deployed for addition, and 1′s and 2′s complement are computed as described below.

Computing 1′s complement

When we apply two input bits a and b to a memristive adder circuit, the output value at two pins of the OCU are $Q_0 = a \oplus b$ and $Q_1 = a \wedge b$, respectively. If we set $a = 1$ then output value at Q_0 will be the complement of the input bit b i.e. $Q_0 = \neg b$. Leveraging this property, 1′ss complement is generated for each bit position. To generate the 2′s complement, the constant 1 is added to the 1′s complement of the input binary number.

Example 4.2.1. *An example is shown in Figure 4.1 where subtraction of* $B = 1001$ *from* $A = 1101$*, both unsigned binary numbers, is performed, i.e., A is the minuend and B*

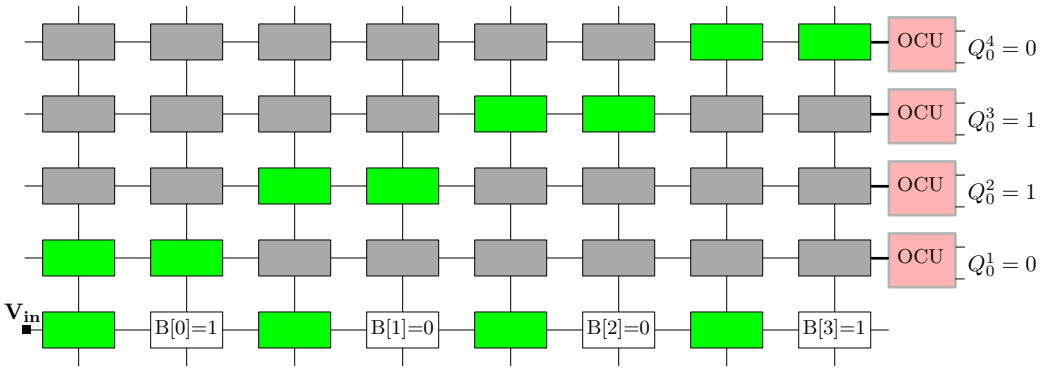


Figure 4.1: Generation of 1′s complement of B, the subtrahend

the subtrahend. In the first cycle, 1′*s complement of* B *is generated in a* 5×8 *crossbar. The input bits B[0], B[1], B[2] and B[3] are written to memristors* M_{12}, M_{14}, M_{16} *and* M_{18}, *respectively. Logic-*1 *is written to memristors* M_{11}, M_{13}, M_{15}, M_{17}, M_{21}, M_{22}, M_{33}, M_{34}, M_{45}, M_{46}, M_{57}, M_{58}. *Logic-*0 *is written in all the other cells.* 1′*s complement is generated at the output pin,* $Q_0^t = \neg B[t]$ *for* $1 \leq t \leq 4$. *In the next cycle, the input number* A, 1′*s complement of* B *and the constant* 1 *are added, as shown in Table 4.1. The final carry being* 1, *is discarded from the result. So* 0100 *is the final result.*

Table 4.1: Adding A to the 1′s complement of B and the constant 1

time t	4	3	2	1	0
A[t]		1	1	0	1
¬B[t]		0	1	1	0
1		0	0	0	1
c[t-1]	1	1	1	1	
SUM	1	0	1	0	0

The computation of 1′s complement is given in Algorithm 5. The block diagram of a memrisor-based subtractor is given in Figure 4.2 which can be used to subtract an unsigned n-bit number $B = b_{n-1} \ldots b_i \ldots b_1 b_0$ from an another unsigned n-bit number

$A = a_{n-1} \ldots a_i \ldots a_1 a_0$.

Algorithm 9: Generation of 1′s complement of an n-bit number

Data: An n-bit unsigned number $B = b_{n-1}...b_2 b_1 b_0$;
A crossbar of size $(n+1) \times 2n$, V_{in}, OCU^2;
Result: $\neg B[t]$, $0 \leq t \leq n-1$;

```
for t=0 to (n − 1) do
    The t^th bit of the input number is written in memristor M_{1,2t+2};
    logic-1 is written in memristor M_{1,2t+1};
    Addition of these two bits is executed in BL_{t+1};
    Q_1^{t+1} = ¬B[t] produces the negation of the t^th bit;
```

Algorithm 10: Crossbar mapping for generation of 1's complement of a n-bit number

Data: AN n-bit unsigned number $B = b_{n-1}...b_2 b_1 b_0$;
A crossbar of size $(n+1) \times 2n$;
n units OCU;
Result: $\neg B[t]$, $0 \leq t \leq n-1$;

```
for t=0 to (n − 1) do
    M_{1,2t+1} ← logic-1;
    M_{1,2t+2} ← B[t];
    for j= 1 to 2 do
        M_{(2+t),(2t+j)} ← logic-1;
logic-0 is written in the rest of the memristors of the crossbar;
WL_1 ← V_in;
for t=0 to n − 1 do
    ¬B[t] = Q_0^1;
```

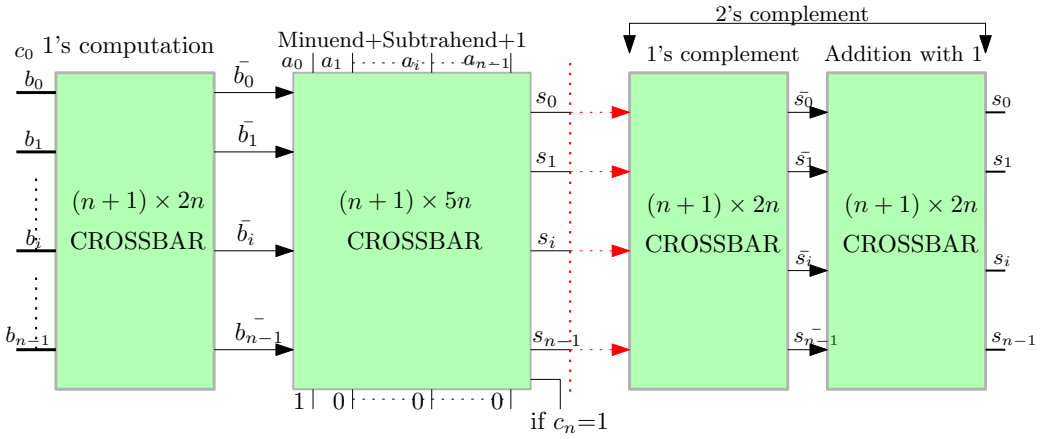


Figure 4.2: Block diagram of an n-bit subtractor

4.2.2 Shift-and-add multiplier

For multiplying two unsigned n-bit numbers, a memristive crossbar-based design is proposed here based on shift-and-add strategy. Two input numbers are written to an $(n+\alpha_n) \times (2n+\alpha_n)$ memristive crossbar in such a way that at each cycle, one output and $\alpha_n - 1$ carry bits are generated.

Example 4.2.2. *An example for multiplying two 4-bits unsigned numbers $A = 1101$ and $B = 1001$ is presented here. The input mapping to the memristive crossbar is shown in Figure 4.3 and output generation at each cycle is given in Table 4.2. When an input voltage is applied at cycle $t = 0$, three output bits are generated where the $Q_0^0 = M[0]$ is the final output bit for that bit position, and $Q_1^0 = c^0[0]$, $Q_2^0 = c^1[0]$ are the carry bits written back to the crossbar to be executed in cycle $t = 1$ and $t = 2$, respectively, as shown in Figure 4.3. The process goes on to $t = 8$ and at the end of the ninth cycle, all output bits are generated.*

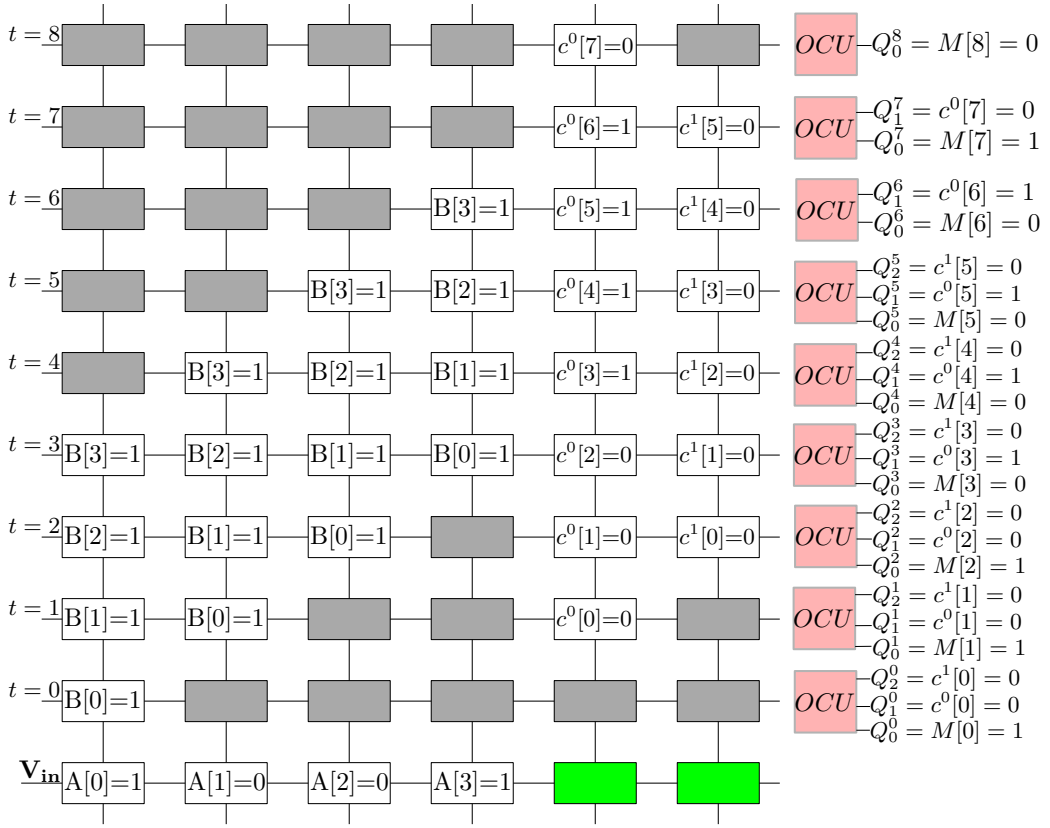


Figure 4.3: A 4-bit multiplier on a 10×6 memristive crossbar

Table 4.2: Execution steps for a 4-bit multiplier

$t=8$	$t=7$	$t=6$	$t=5$	$t=4$	$t=3$	$t=2$	$t=1$	$t=0$
					A[0]B[3]	A[0]B[2]	A[0]B[1]	A[0]B[0]
				A[1]B[3]	A[1]B[2]	A[1]B[1]	A[1]B[0]	
			A[2]B[3]	A[2]B[2]	A[2]B[1]	A[2]B[0]		
		A[3]B[3]	A[3]B[2]	A[3]B[1]	A[3]B[0]			
$c^0[7]$	$c^0[6]$	$c^0[5]$	$c^0[4]$	$c^0[3]$	$c^0[2]$	$c^0[1]$	$c^0[0]$	
	$c^1[5]$	$c^1[4]$	$c^1[3]$	$c^1[2]$	$c^1[1]$	$c^1[0]$		
M[8]	$c^0[7]$**M[7]**	$c^0[6]$**M[6]**	$c^1[5]c^0[5]$**M[5]**	$c^1[4]c^0[4]$**M[4]**	$c^1[3]c^0[3]$**M[3]**	$c^1[2]c^0[2]$**M[2]**	$c^1[1]c^0[1]$**M[1]**	$c^1[0]c^0[0]$**M[0]**

Algorithm 11 explains the implementation of a shift-and-add multiplication of two n-bit unsigned binary numbers $A[n]$ and $B[n]$ on a memristor-based hybrid crossbar of size $(2n+\alpha_n)\times(n+\alpha_n)$ employing CIM. The input number A is written to the first n memristors

Algorithm 11: Memristive crossbar based multiplication of two n-bit unsigned numbers

Data: Two n-bit unsigned numbers $A[n]$ and $B[n]$;
A crossbar of size $(2n + \alpha_n) \times (n + \alpha_n)$, V_{in}, OCU^{α_n};
Result: Final output product bits $M[t]$ where $0 \leq t \leq (2(n-1) + \alpha_n)$;

1 The multiplicand input A is written into memristors $M_{1,1}$ to $M_{1,t}$ s.t. $A[t]$ is written into $M_{1,t}$;
2 *Logic*-1 is written into memristors $M_{1,t+1}$ to $M_{1,(n+\alpha_n)}$;
3 The multiplier input B is written along the each BL starting from BL_1 to BL_n s.t. along BL_k, $B[t]$ is written into memristor $M_{(k+t+1),k}$;
4 **for** $t = 1$ *to* $2n + \alpha_n$ **do**
5 At t^{th} cycle, the output is generated at WL_{t+2} where Q_0 is the final output bit, i.e., $Q_0^{t+1} = M[t]$;
6 Rest of the output bits are carry bits to be written back to the memristors for future execution s.t. Q_k^{t+1} is used in cycle $k + t + 1$;

connected to the WL_1, and the other number B is repeatedly written on the n BLs of the crossbar starting from BL_1 to BL_n with one row of upward shift. In the crossbar, the last α_n BLs are reserved for applying the carry bits produced in each cycle. In this context, Algorithm 12 demonstrates the precise mapping of input bits and generated carry bits to the corresponding memristor cells within the crossbar structure. During each iteration, one final product bit is generated along with α_n carry bits to be utilized for future iterations, i.e., at t^{th} cycle, the output is generated at OCU connected to WL_{t+2} where $Q_0^t = M[t]$ and the remaining $(n + \alpha_n)$ bits are carry bits. The peripheral control circuit writes the carry bits to the appropriate memristors connected to BLs reserved for the carry bits for future iterations. The complete multiplication procedure takes $(2n + \alpha_n)$ cycles. A block diagram is shown in Figure 4.4 where multiplication is performed for two unsigned n-bit binary numbers. Table 4.3 shows the required crossbar size, OCU size, and the number of cycles required for executing the multiplication of binary numbers with different word-size.

4.3 Logical Operations

Logical operations such as NOT, AND, and OR can be implemented on a hybrid memristive crossbar for enabling CIM. In the proposed method, AND and NOT operations would require a single cycle whereas OR can be executed multiple cycles.

Algorithm 12: Crossbar mapping for the memristive Crossbar based multiplication of two n-bit unsigned numbers

Data: Two n-bit unsigned numbers
$A = a_{n-1} \dots a_2 a_1 a_0$ and
$B = b_{n-1} \dots b_2 b_1 b_0$;
A crossbar of size $(2n + \lfloor \log_2(n + \log_2 n) \rfloor) \times (n + \lfloor \log_2(n + \log_2 n) \rfloor)$;
Result: Final output product bits M;

```
for i = 1 to n do
    M_{1,i} ← A[i − 1];
    for j ← 1 to n do
        M_{j+i,i} ← B[j − 1];
for k = (n + 1) to (n + ⌊log2(n + log2 n)⌋) do
    M_{1,j} ← logic-1;
t = 1;
while t ≤ (2n + ⌊log2(n + log2 n)⌋) do
    for k = 1 to (⌊log2(n + log2 n)⌋) do
        M_{(t+2),(n+k)} ← c^{k−1}[t − k];% feed carry bits from addition of partial products using
          bit-parallel RCA
    WL_1 ← V_in;
    M[t] = Q_0^t;
    for l = 1 to ⌊log2(n + log2 n)⌋ do
        c^{l−1}[t] = Q_l^t;
    t = t + 1
```

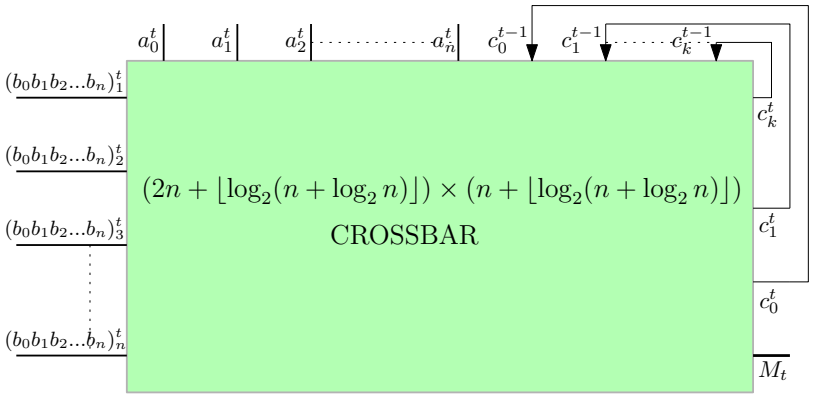


Figure 4.4: Block diagram of a multiplier for two n-bit numbers

Table 4.3: Crossbar size and the number of cycles for multiplying two n-bit unsigned numbers

Word size n	Crossbar size $(2n+\alpha_n) \times (n+\alpha_n)$	#OCU pin $(n+\alpha_n)$	#Cycle $(2n+\alpha_n)-1$
2	3×5	3	5
4	6×10	6	10
8	11×19	11	19
16	20×36	20	36
32	37×69	37	69
64	70×134	70	134
128	135×263	135	263

4.3.1 NOT-operation

NOT is implemented on a 2×2 crossbar where one input bit a is used to set memristor M_{11} and all other memristors are set to *logic*-1. When an input voltage is applied to WL_1, $Q_1 = \neg a$ is observed at the output of an OCU having two output pins. Using the same procedure, word-level NOT-operation can be performed in one pass based on the available crossbar size.

Example 4.3.1. *The mechanism of word-level NOT-operation of on* $B = 1001$ *is illustrated in Figure 4.5.*

4.3.2 AND-operation

Two separate techniques are proposed for implementing multi-bit AND-operation on a memristive hybrid crossbar. One is single-cycle execution where all bits of the inputs are written in the crossbar and the final output is generated. However, it would need a large-size crossbar. The other one is executed in multiple cycles but consumes requires crossbar area.

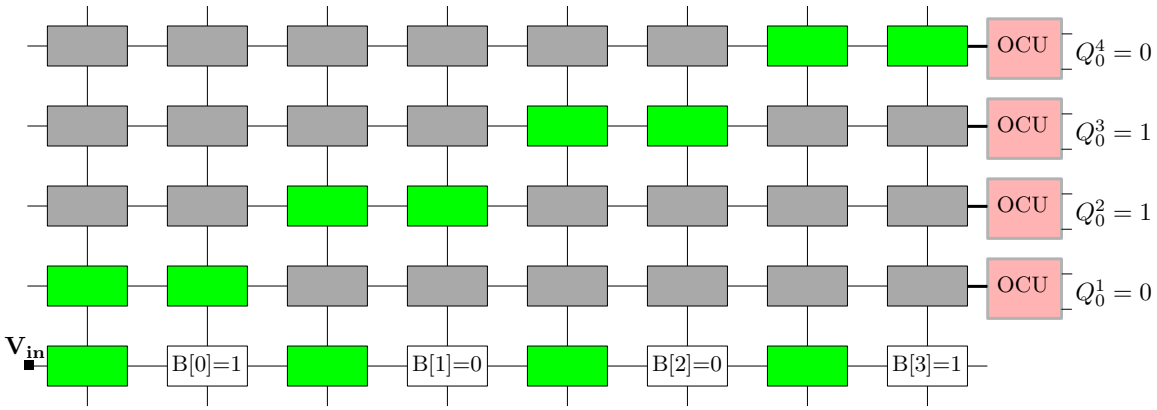


Figure 4.5: Word-level NOT-operation on $B = 1001$

Single-cycle implementation

We consider an AND operation on n binary bits where $2^{i-1} \leq n < 2^i$, *and* $i \in \mathbb{N}$. A crossbar of size 2×2^i is required to complete the operation in one cycle. Among all the memristors connected to WL_1, n input bits are written to the first n memristors and *logic*-1 is set to the rest of the memristors. An OCU having $i + 1$ output pins is connected to WL_2 where the MSB pin (Q_i) provides the final output.

Example 4.3.2. *Here two examples are present for executing AND operation in hybrid memristive crossbar.*

1. $y_1 = 1 \wedge 0 \wedge 1$
2. $y_2 = 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1$

The output y_1 is generated on a 2×4 crossbar as shown in Figure 4.6(a) and the output y_2 is generated on a 2×8 crossbar as shown in Figure 4.6(b), respectively.

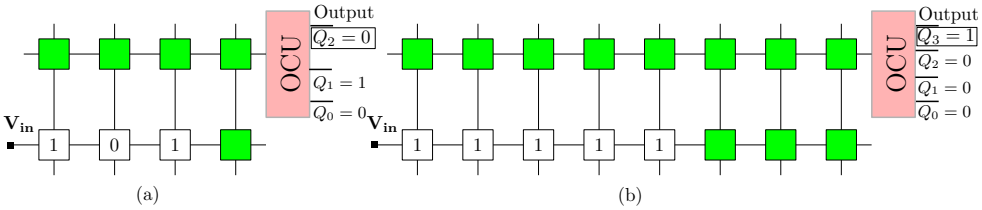


Figure 4.6: Implementing bitwise AND-operation for (a) the word $y_1 = 101$ with output $Q_2 = 0$ and (b) the word $y_2 = 1111$ with output $Q_3 = 1$

Multiple-cycle implementation

A multi-cycle AND module is proposed for optimizing the size of the memristive crossbar. In order to execute AND operation on n binary bits, the idea is to represent n as a sum of power of 2 and execute each of them in a single cycle. In the next cycle, the output bits of the previous cycle are executed. This process is continued till the operation is completed.

Example 4.3.3. *Consider AND-operation on 13-bits. Since* $13 = 2^3+2^2+2^0$*, three distinct AND operations are realised on a crossbar of size* 2×7 *(*$2^{3-1}+2^{2-1}+2^0$*). In the next cycle, the three generated output bits are again inserted into an* 2×2 *zone of the same crossbar to generate the final output bit as shown in Figure 4.7.*

Depending on the value of n, single- or multi-cycle implementation of AND-operation would be convenient as described below. Case-1 (Case-2) will be suitable for single-(multi-) cycle realization.
Case-1: $2^{i-1} < n \leq 2^i$ and $2^i - n \leq i$ where $i \in \mathbb{N}$
Case-2: $2^{i-1} < n < 2^i$ and $2^i - n > i$ where $i \in \mathbb{N}$

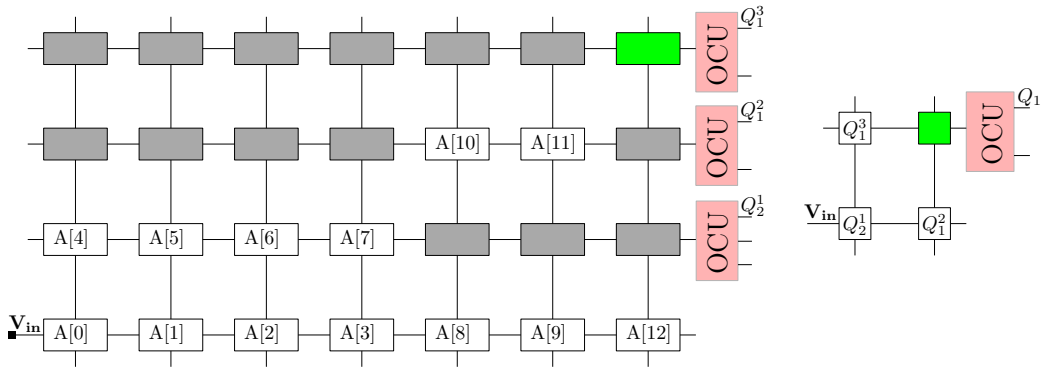


Figure 4.7: Implementation of 13-bit wide AND-operation

4.3.3 OR-Operation

Multi-bit OR-operation can be performed by applying a sequence of NOT-, AND-, and NOT-operation. In order to reduce complexity, this can be done following a sequence: Binary-addition, NOT, AND, NOT. An example with 6 bits is illustrated in Figure 4.8.

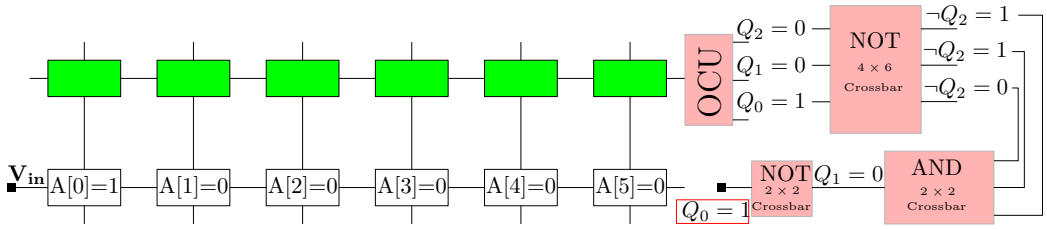


Figure 4.8: Implementation of k-bit OR-operation on a memristive crossbar

Table 4.4: Crossbar size and the number of cycles for implementing logical operations

#bits	NOT gate		AND gate		OR gate	
n	Crossbar size	#cycles	Crossbar size	#cycles	Crossbar size	#cycles
1	1×1	1	NA	NA	NA	NA
2	3×4	1	1×1	1	2×2	4
3	4×6	1	2×2	1	3×4	4
4	5×8	1	2×2	1	4×6	4
8	9×16	1	4×4	1	5×8	4
13	14×26	1	4×7	2	5×8	4
15	16×30	1	4×8	2	5×8	4
24	25×48	1	3×12	2	6×10	4
31	32×62	1	5×16	2	6×10	4
63	64×126	1	6×32	3	7×12	4

4.4 Simulation Results and Discussion

4.4.1 Results

Table 4.4 provides the crossbar size and execution cycles required for performing three logical operations: NOT, AND, and OR. Table 4.5 provides a comprehensive overview of the utilized crossbar, including its crossbar-size and the number of memristor cells required for writing the *input bits*, *logic-1* bits, and *logic-0* bits. Additionally, the table presents the number of execution cycles needed for the proposed arithmetic and logic operation. In Table 4.6, we compare the results of our method with previous work. For implementing

Table 4.5: Crossbar size, number of memristors used and execution cycles for the proposed ALU designs

Operation	Step	Crossbar size	#Memristors for writing			#cycles
			input bits	*logic*-1	*logic*-0	
4-bit CSA inspired addition		2×3	3	3	0	6
4-bit CLA inspired addition	$P(t)$ & $G(t)$ generation	5×8	8	8	24	1
	carry generation	7×9	15	6	42	1
	final addition	5×12	12	12	36	1
4-bit subtraction	1's complement	5×8	4	12	24	1
	addition	2×3	3	3	0	6
4-bit multiplication		10×6	34	2	24	1
4-bit NOT operation		5×8	4	12	24	1
4-bit AND operation		2×2	4	0	0	1
4-bit OR operation	addition	2×4	4	4	0	1
	NOT operation	4×6	3	9	12	1
	AND operation	2×2	3	1	0	1

Table 4.6: Comparison of number of memristors and execution cycles for our designs vs. prior works

Operation	(TGY+17),(YT18)		(GS17a)		(BS14)		**Proposed method**	
	#mems	#cycles	#mems	#cycles	#mems	#cycles	#mems	#cycles
Full Adder	15	13	N.A.	N.A.	N.A.	19	8	2
4-bit CSA	57	49	N.A.	N.A.	N.A.	44	24	4
4-bit CLA	109	101	N.A.	N.A.	N.A.	38	119	3
n-bit CSA	$14n+1$	$12n+1$	N.A.	N.A.	N.A.	N.A.	$6n$	n
n-bit CLA	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	$7(n^2+1)$	3
4-bit Multiplier	N.A.	N.A.	29	116	N.A.	127	30	9
n-bit Multiplier	N.A.	N.A.	$7n+1$	$2n^2+21$	N.A.	N.A.	$(2n+\alpha_n) \times (\alpha_n+1)$	$2n+\alpha_n-1$

an in-memory module, the size of the available crossbar is very important. To the best of our knowledge, no previous attempt was made to scale the module size based on available memory. In our design, the size of the module can be optimized depending on the task schedule and available memory size.

We define the step current $I_\Delta = V_{in}/2M_{on}$, the amount of current passing through an active path. When multiple active paths are present in the crossbar, the output current is the integer multiple of I_Δ. The value of I_Δ depends only on the input voltage v_{in}, and the ON-resistance M_{on} of the memristor. We perform resistive spice simulation using LTSpice with $V = 5V$, $M_{on} = 100\Omega$ and $M_{off} = 100k\Omega$. We measure the sum of current-values passing through five parallel paths in a 5×5 crossbar, as shown in Figure 4.9, where $I_\Delta = 25mA$. The four other values of current are consecutive integer multiples of I_Δ. As $M_{off} >> M_{on}$, the current passing through the OFF-memristor (I_{OFF}) is negligible in comparison to I_Δ, so the effect of I_{OFF} is not considered while calculating I_Δ. Further, we observed that the value of I_Δ is nearly constant when $\frac{M_{on}}{M_{off}}$ is very small ($< 10^{-3}$), but becomes unstable as $\frac{M_{on}}{M_{off}}$ increases significantly.

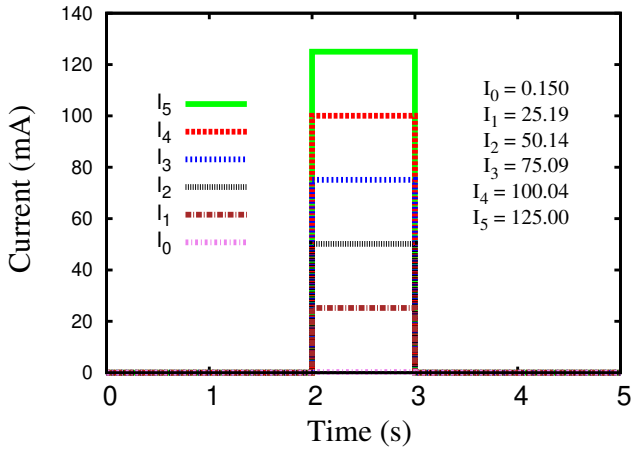


Figure 4.9: Simulation results with five discrete current values

4.4.2 Discussion

Although previously proposed methods used different implementation strategies with memristors, we present a comparative study of crossbar-size and execution time needed for various arithmetic and logical operations. The method proposed in (TGY+17) used MAGIC-

based design to implement the operations whereas in (GS17a), the authors have implemented IMPLY- and MAD-based design. IMPLY-based synthesis suffers from difficulties in multi-step operations and interference. MAGIC-based design often leads to high latency and area. A set of mMPU instructions, including FILL and MCMP instructions, to execute various set operations (union, cartesian product, transitive closure, and power set generation) on the memristor crossbar is proposed in (KP23). The energy savings achieved through in-memory set operations are significant. However, the delay of in-memory comparison and other ALU operations is high. In our work, we perform all computations in analog domain in order to speed up in execution, and the final output is converted to digital domain. For the memristor count, we have also included those needed to save the final output of the module. Our method reduces not only the memristor count but also the number of time cycles significantly. Even though some operations discussed in Section 3.3.3 and 4.3.3 may not perform as efficiently as others, we have included them for completeness of in-memory computation.

4.5 Summary

Implementations of several ALU and logical modules have been proposed that are suitable for in-memory computation on a hybrid-memristor crossbar where each input bit is mapped to a memristor-state.. Computation is executed in the analog domain by generating output current. A simple CMOS-based peripheral circuit *OCU* consisting of a current comparator and an *ADC* is employed that checks the output current and maps it to the corresponding output logic. While determining the output current the wire resistance is not considered as the length of each current-conducting path is the same. The proposed methods are fast as all computations are performed in the analog domain by leveraging the intrinsic nature of the memristor. In the proposed method, the ratio always generates integer values which are then converted into binary numbers by ADC. However, in practice, round-off errors may arise and additional circuits may be required to mitigate those. The method is capable of handling only unsigned integers. While executing operations with large inputs, the generated outputs may sometime exceed the crossbar space. In order to tackle such cases, the inputs are partitioned into smaller-size groups so as to fit into the crossbar size. The number of execution cycles will thus increase accordingly. Also, memory overflow has not been addressed in this work.

CHAPTER

FIVE

TEST OPTIMIZATION IN FULL MEMRISTIVE CROSSBARS

Contents

5.1 Introduction

In recent times, memristors have demonstrated great potential in the development of memory and logic subsystems. Utilizing memristor arrays, a 2D-crossbar architecture offers a practical solution for storing multi-valued memory states, capitalizing on the analog variation of current-induced resistance within these cells. The integration of CMOS components with non-CMOS memristor cells expands their applicability to diverse and intricate system designs. Nevertheless, it's important to note that present-day resistive random access memory (RRAM) is susceptible to manufacturing defects and sensitive to operational modalities due to its inherent structure. The high integration density of RRAM elements may induce resistive shorts or opens like in static random access memory (SRAM) or dynamic random access memory (DRAM). The behavior of RRAM elements in presence of such defects needs robust testing to determine the correctness level of such a memory.Existing techniques for testing memristor arrays are either ad-hoc in nature or suited for application-specific designs. These techniques consumes huge time to test a large sized crossbar and very few of them have shown concern for optimizing the test time.

In this chapter, we propose a network-based approach to analyze a 2D memristive crossbar and identify fault-sensitization paths. Specifically, our method optimizes test time for full-size square and rectangular memristive crossbars using a path-based technique guided by maximum matching in bipartite graphs. Through simulation results with LTspice, we showcase the effectiveness and superiority of our approach compared to prior methods in terms of test time and fault-coverage.

5.2 Preliminaries

5.2.1 Sneak-path

Since memristors are bi-directional devices, while accessing a particular memristor in a crossbar architecture, a small amount of current may also flow through some un-selected memristor cells. This current is known as sneak-current, and the path through which it flows is called a sneak-path (see Figure 5.1). In an ideal functional mode the current flows from the source line to the ground line passing through only the desired memristor cell at the intersection between the activated WL and BL. Unfortunately in most of the practical applications the sneak current flows through many sneak paths beside the desired one. These paths act as an unknown parallel resistance to the desired cell resistance. Therefore,

the actual output current differs from that of the desired one which causes errors in output reading. The added resistance of the sneak paths also significantly narrows the noise margin and reduces the maximum possible size of a memristor array. In the functional mode, the activation of sneak-paths is undesirable and various techniques are used to mitigate the sneak path current as listed below.

- Multistage reading
- Unfolded architecture
- Diode gating
- Transistor gating
- Complimentary memristors
- Using memristors nonlinearity
- AC sense

Among all the proposed solutions mentioned above for mitigating the sneak paths, the most popular solution is to to add a diode or a transistor to each memory cell, producing a new cell of one diode and one memristor ($1D1M$) or one transistor and one memristor ($1T1M$), respectively. Although, Such a arrangement would eliminate sneak paths but this will ruin the high memristor-memory density, since the gating transistor's size is much larger than that of the memristor and also increase delay of the signal. In the proposed work, an $1T1M$ unfolded crossbar architecture is used for test optimization.

5.2.2 Fault model

We consider a memristive crossbar where each memristor element has two memristance states. The general convention is that the high memristance state of a memristor is defined as OFF state and denoted as *logic*-0, and the low memristance state of a memristor is defined as ON state and denoted as *logic*-1, respectively. A fault said to be occurred when the observed output differs from the expected output. Different types of fault may occur due to hetero-structure fabrication process and parametric variations (KRKS13a) (GPM09a), which can be broadly categorized into two classes as given below (BA13).

1. *Self-fault*: If a memristor $M_{i,j}$ is targeted for sensitization and fault occur in the same memristor element then the fault is called as self-fault.
2. *Coupling-fault*: If a memristor $M_{i,j}$ is targeted for sensitization and fault occur in some different memristor $M_{k,l}$ for the effect of sensitization then the fault is called as coupling-fault.

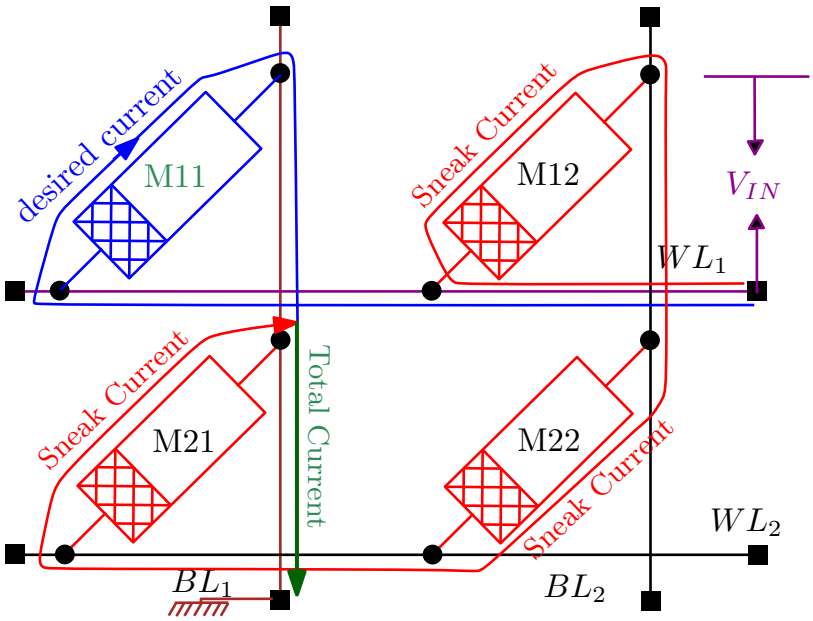


Figure 5.1: Desired current (green line), sneak-current (red line) and total current (blue line) in a 4×4 crossbar architecture when input voltage is applied across memristor M_{11}.

Reading *logic*-0 and *logic*-1 from the memristors, and writing *logic*-0 and *logic*-1 into the memristors are four basic operations denoted by $\{w0, w1\}$ and $\{r0, r1\}$, respectively. These Four standard memristor-access operations are required in order to detect all *self-faults*, described below:

- *Stuck-at*-0 *(SA0)*: The $SA0$ fault is said to occur when the output of a memristor is always logic-0 irrespective of the voltage applied across it. One of the major causes behind the $SA0$ fault is the deficiency of doping density in a memristor. When there is an "open" circuit in any of the WLs, BLs, or at the crosspoint, $SA0$ faults may also occur. In order to detect $SA0$ fault in a memristor, the memristor cell is sensitized with logic-1 when the memristor state is set to logic-0. This fault is represented by $\langle 1/0 \rangle$ which denotes logic-1 is the required state while logic-0 is the actual output state.
- *Stuck-at*-1 *(SA1)*: If $SA1$ occurs, the output of a memristor is permanently logic-1 regardless of the voltage applied. It generally happen due to excessive doping concentration in the memristor. In opposite to $SA0$ faults, an $SA1$ fault can also occur when the WLs or BLs are shorted to V_{dd}. In order to detect $SA1$ fault in a memristor, the memristor cell is sensitized with logic-0 when the memristor state is

set to logic-1. This fault is represented by $\langle 0/1 \rangle$.

- *Slow-write*-1 *(SW1)*: When a single voltage-pulse for performing write operation is inadequate to change a memristor from logic-0 to logic-1 state, the fault in called $SW1$. Insufficiency in doping concentration causes this kind fault in the memristor. In place of single excitation, a sequence of excitation ($\{w0, w1, r1\}$) are required to detect $SW1$ fault. In order to detect an $SW1$ fault, first two consecutive write operations of logic-0 and logic-1 is performed, and then the memristor is sensitized with logic-1.
- *Slow-write*-0 *(SW0)*: Similarly, in case of $SW0$ faults, a single voltage-pulse for performing write operation is inadequate to change a memristor from logic-1 to logic-0 state. This transition is slow due to the excessive doping concentration in memristor cell. In order to detect an $SW0$ fault, the sequence $\{w1, w0, r0\}$ is required.
- *Deep*-0 *(D0)*: An increase in the length or a decrease in the cross-section of a memristor results in $D0$ fault. This physical defect causes a shift in the upper (R_{off}) resistance limits of the memristor by a constant value δ. In presence of $D0$ fault, the new upper limit of the resiatance is $R_{off} + \delta$. Therefore, a logic-1 write pulse, determined for fault free memristor is unable to change state from OFF to ON. The sequence $\{w0, w0, w1, r1\}$ is applied to detect $D0$ fault.
- *Deep*-1 *(D1)*: An decrease in the length or a increase in the cross-section of a memristor results in $D1$ fault. This physical defect causes a shift in the lower resistance (R_{on}) limits of the memristor by a constant value δ. In presence of $D1$ fault, the new upper limit of the resiatance is $R_{on} + \delta$. Therefore, a logic-0 write pulse, determined for fault free memristor is unable to change state from ON to OFF. The sequence $\{w1, w1, w0, r0\}$ is applied to detect $D1$ fault.

Note that the detection of hybrid faults (i.e., multiple types of faults occurring simultaneously) is very complex, as they may impact crossbar-functionality in different fashions. The proposed method is capable of handling multiple faults of the same type only. The path-selection techniques described in our work are invariant to the location of faults in the crossbar.

5.3 Problem Formulation

We consider $1T1M$ crossbar architecture in order to conveniently sensitize either a single or a set of memristors for testing. A memristor M_{ij} is said to be accessed if a voltage source is applied between word line WL_i and bit line BL_j. Therefore, when we access a

specific memristor in an $m \times n$ crossbar, we get a corresponding memristor network comprising of all possible sneak-paths (hereafter referred to as *sneak_path_network*). Due to the symmetric nature of the crossbar, the structure of all the mn *sneak_path_network*s generated from an $m \times n$ crossbar is invariable with respect to the selection of the memristor only the position of the memristors differs from structure to structure. In Figure 5.2(a) and 5.2(b), we have shown (only the memristors) a 4×4 $1T1M$ crossbar and its corresponding *sneak_path_network* while accessing memristor M_{11}.

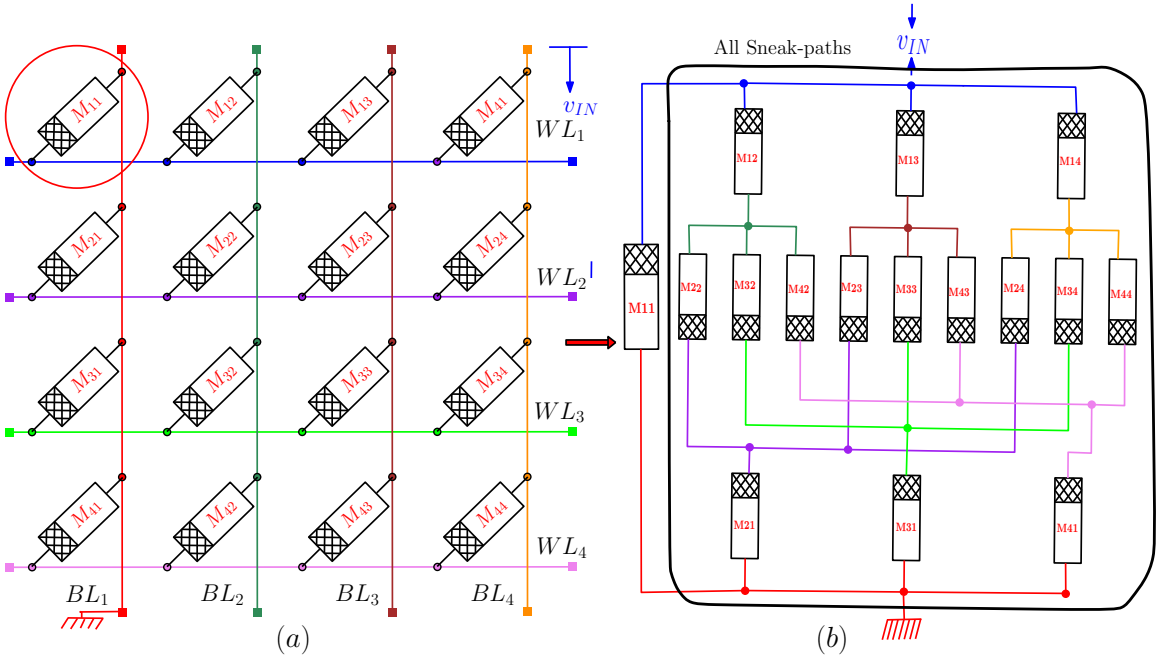


Figure 5.2: (a) $M11$ is accessed in a 4×4 crossbar (b) corresponding *sneak_path_network*.

Current is the key parameter for memristor-based crossbar testing. Current collects important information while passing through the internal cells of the memristor-based crossbar. In the proposed architecture, we purposefully activate a few signal paths through the memristive crossbar (hereafter referred to as sneak-paths) to facilitate testing. By controlling individual memristors of the unfolded crossbar and by using a special decoder, one can select multiple rows and columns in order to activate a desired path through it. Single or multiple sneak-paths have to be suitably selected out of the large number of possible paths from the source line to the ground line so that the current passing through these serve as an indicator of the presence of a fault, if any. The paths should be so chosen that I_{ideal}, the

nominal sneak current differs notably from I_{faulty} in the presence of a fault. The values of I_{ideal} and I_{faulty} are compared in order to detect a potential fault present in the crossbar. In the *sneak_path_network*, corresponding to an $n \times n$ crossbar, the length l_p of a selected sneak-path is the number of memristors present in it. The length of all these paths lies between 3 and $2n-1$, but in order to test a path the maximum and minimum allowable path lengths $(l_p)_{max}$ and $(l_p)_{min}$ respectively, depend on the memristor parameters, fault types, applied voltage, and the resolution of the current sense-amplifier.

5.3.1 Problem statement for the sneak-path network

In order to detect $SA0$ or $SA1$ faults, all the memristors on the selected path/paths are sensitized with *logic*-1 (M_{on}) or *logic*-0 (M_{off}), respectively. Let I_{th} and I_{sat} be the minimum and maximum current value of the current-sense amplifier. If k_1 and k_2 be the path length for detecting $SA0$ or $SA1$ faults then $I_{th} \leq \frac{V_{in}}{k_1 * M_{on}} \leq I_{sat}$ and $I_{th} \leq \frac{V_{in}}{k_2 * M_{off}} \leq I_{sat}$ where V_{in} is the input voltage. As $M_{on} << M_{off}$, $k_2 << k_1$, long single path is preferred for $SA0$ testing while multiple short paths are preferred for $SA1$ testing. In our work, we assume that $(l_p)_{max}$=$(l_p)_{min}$=l_p then the problem statement for single path testing as well as multiple paths testing is as given below.

Single path detection: *Given a sneak_path_network and path-length l_p, determine the minimum number of paths from source to the ground line in the network such that the length of each path is equal to l_p, and these collectively cover all the edges.*

Multiple paths detection: *Given a sneak_path_network, path-length l_p and an integer k, determine the minimum number of path sets where each set consists of k parallel paths from source to the ground line in the network such that the length of each path is equal to l_p, and this path sets collectively cover all the edges.*

Finding a single path or multiple paths in a *sneak_path_network* is a difficult task. In order to circumvent it, we have converted a *sneak_path_network* into an equivalent graph and by utilizing well-known graph algorithms, we have identified the desired sneak-paths in the network which are to be sensitized.

5.3.2 A graph-based formulation

Given a *sneak_path_network* shown in Figure 5.2(b), we construct a graph $G_{spn} = (V, E)$ where there is a vertex v for each of the BLs and the WLs, and an edge (v_i, v_j) in E for each memristor connecting a WL to a BL in the network.

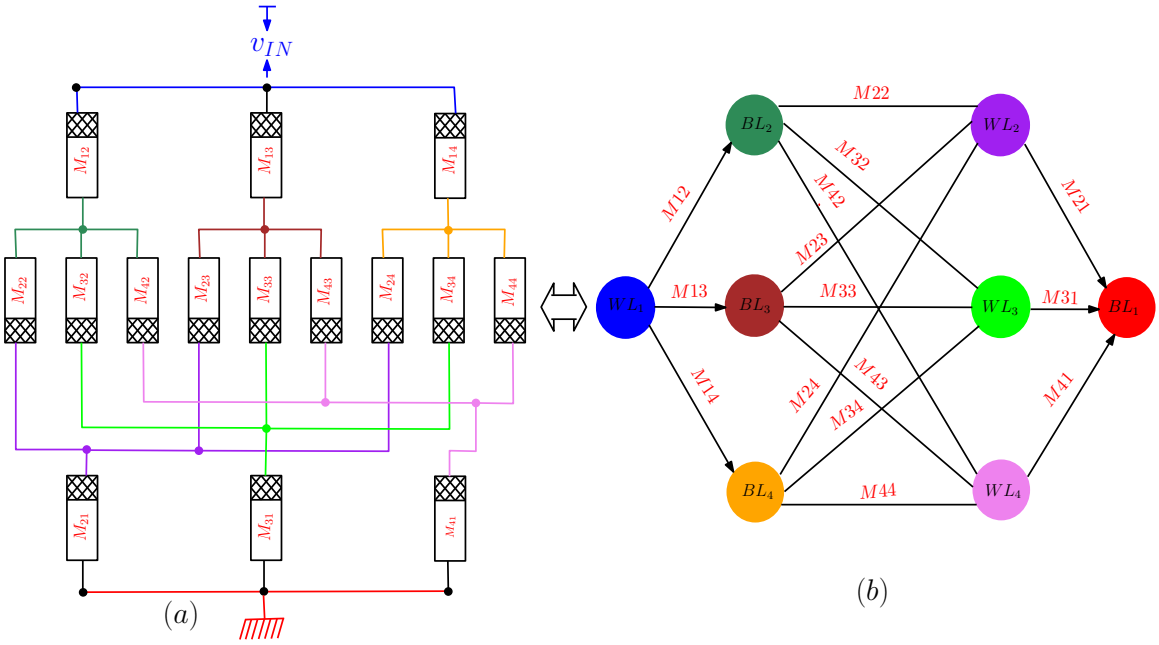


Figure 5.3: (a) *Sneak_path_network* and (b) its corresponding graph G_{spn}.

The graph $G_{spn}(V, E)$ contains a source vertex s and a destination vertex d representing the WL connected to the source voltage, v_{IN}, and the BL connected to the ground, respectively. Vertex sets containing all the vertices adjacent to s and d are named as L and R, respectively, such that $V = L \cup R \cup \{s, d\}$. The $G_{spn}(V, E)$ defined above is used for our all future analysis. A *sneak_path_network* generated from a 4×4 crossbar (shown in Figure 5.2) and its corresponding graph $G_{spn}(V, E)$ are shown in Figure 5.3(a) and Figure 5.3(b), respectively. therefore, the test-time reduction for a given memristor-based crossbar array can be formulated as the path selection in the graph G_{spn} as follows:

Test Optimization by Single Path Selection (TOSPS): *Given a path-length l_p, determine the minimum number of paths ν in the graph $G_{spn}(V, E)$ from source vertex s to destination vertex d such that the length of each path is equal to l_p, and these paths collectively cover all the edges in E.*

Test Optimization by Multiple Paths Selection (TOMPS): *Given a path-length l_p and an integer k, determine the minimum number ν of path sets in its graph G_{spn}, where each path set contains k disjoint parallel paths from source vertex s to destination vertex d such that the length of each path is equal to l_p, and these ν path sets collectively cover all*

the edges in E.

5.4 Testing of Full Square Crossbars

5.4.1 Graph decomposition

In order to solve TOSPS and TOMPS, we first decompose $G_{spn}(V, E)$ into three sub-graphs as follows.

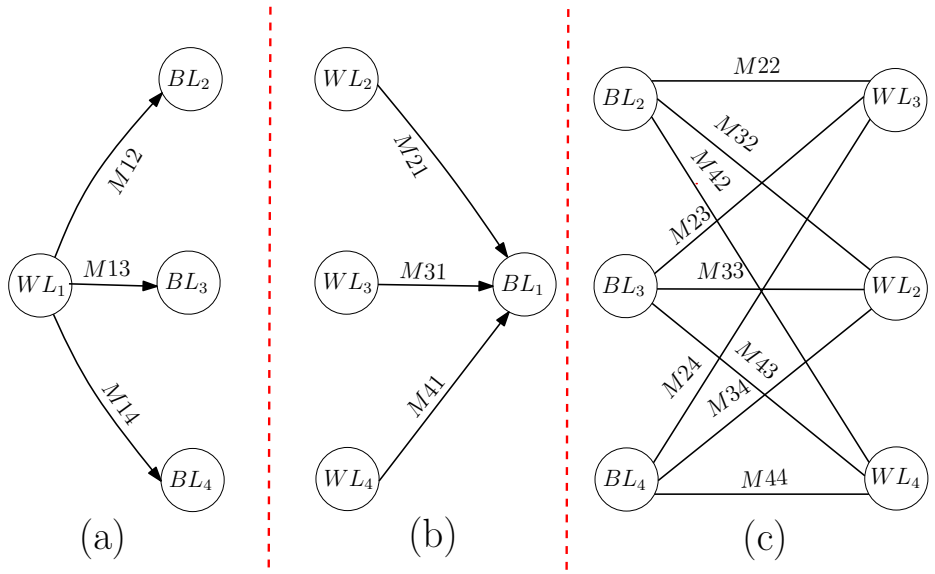


Figure 5.4: Decomposition of $G_{spn}(V, E)$ in Fig. 5.3(b): (a) $G_{bds}(V_s, E_s)$ (b) $G_{bdd}(V_d, E_d)$ (c) $G_{icb}(V_i, E_i)$.

Boundary Directed Source Graph ($G_{bds}(V_s, E_s)$): This sub-graph contains the source vertex s and all the vertices in set L and all the edges incident at vertex s. $G_{bds} = (V_s, E_s)$ where $V_s = \{s\} \cup L$ and $E_s = \{< s, l_i > \mid l_i \in L\}$ (Figure 5.4(a)).

Boundary Directed Destination Graph ($G_{bdd}(V_d, E_d)$): The destination vertex d and all the vertices in set R and all the edges incident at vertex d forms this sub-graph. $G_{bdd} = (V_d, E_d)$ where $V_d = R \cup \{d\}$ and $E_d = \{< r_j, d > \mid r_j \in R\}$ (Figure 5.4(b)).

Intermediate Complete Bipartite Graph ($G_{icb}(V_i, E_i)$): All the vertices in set $L \cup R$ along with the associated edges form this sub-graph. This graph is a complete bipartite graph and most of the edges of $G_{spn}(V, E)$ is contained by this graph. $G_{icb} =$

(V_i, E_i) where $V_i = L \cup R$ and $E_i = \{(l_i, r_j) |\ l_i \in L, r_j \in R\}$ (Figure 5.4(c)).

The decomposition of the graph $G_{spn}(V, E)$ is done in such a way that there is no common edge among $G_{bds}(V_s, E_s)$, $G_{bdd}(V_d, E_d)$ and $G_{icb}(V_i, E_i)$ i.e. $E_s \cap E_d \cap E_i$. In our path construction algorithm, in order to select a path og length l_p exactly one edge each from G_{bds} and G_{bdd} is selected in the solution path. The rest of the $(l_p - 2)$ edges come from G_{icb}. Therefore our path selection in G_{spn} is equivalence to find long edge-disjoint path/paths in G_{icb} efficiently.

5.4.2 Square crossbar testing

The graph $G_{spn}(V, E)$ for an $n \times n$ crossbar has $|V| = 2n$ and $|E| = n^2 - 1$; note that the edge corresponding to the memristor across which the input voltage is applied, is not included in the *sneak_path_network* and is absent in G_{spn}. Here we propose matching based algorithms for selecting paths of length l_p and based on the value of l_p two cases arise as given below.

Case Q1: $l_p \geq n - 1$

In a selected path of length l_p, there are $(l_p - 2)$ edges from graph G_{icb} and one edge each from G_{bds} and G_{bdd}. The longest path in G_{spn} has length $l_p = (2n - 1)$. If the length of each selected path is $2n-1$, then the number of paths required to cover all the edges is at least $\nu = \frac{n^2-1}{2n-1} = \lceil \frac{n}{2} \rceil$. However, no two edges of $G_{bds}(V_s, E_s)$ as well as of $G_{bdd}(V_d, E_d)$ can appear on the selected path. As $|E_s| = |E_d| = (n - 1)$, there must be at least $(n - 1)$ paths to cover all the edges of G_{spn}. Therefore, we restrict the length l_p of a path to $n - 1$ and aim to find $n - 1$ paths, each of length $n - 1$, in G_{spn}. Since G_{icb} is the complete bipartite graph $K_{n-1,n-1}$, we have $n - 1$ perfect matchings and each matching has $n - 1$ edges.

In order to construct paths from matchings in G_{icb}, the pertinent question is whether any set of distinct perfect matchings covering all the edges of G_{icb} are suitable for finding the paths in $\mathcal{P}_{n-1}$. In Figure 5.7 we demonstrate a negative answer to this question with an example for $K_{4,4}$ where a cycle is formed when the four perfect matchings are concatenated according to Algorithm 13.

Finding valid matching sets using a Latin Square

A Latin square is an $(n \times n)$ matrix with elements from a set of n distinct symbols such that each symbol appears exactly once in each row and in each column (KD15). Each entry is written as a triple (r, c, s), where r is its row index, c its column index, and s is the symbol.

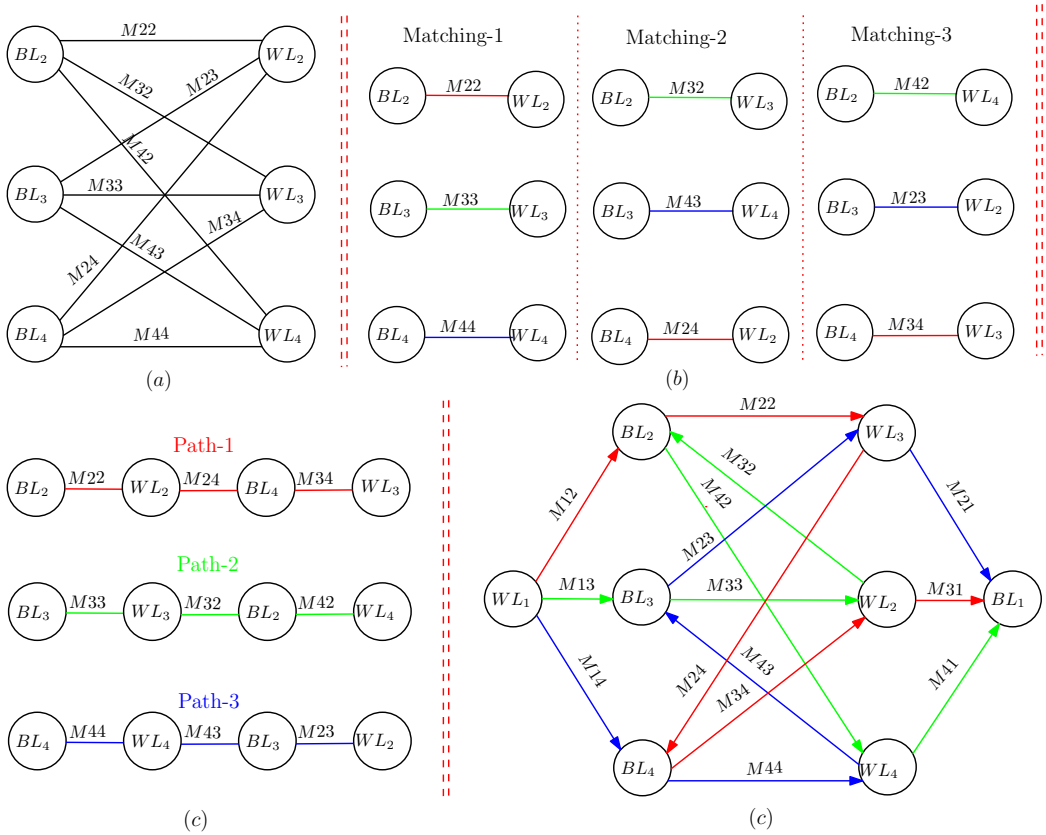


Figure 5.5: (a) $K_{3,3}$ G_{icb} and (b) a cover with three edge-disjoint perfect-matching (c) the three paths in G_{icb}, and (b) three complete sneak-paths in G_{snp} indicated by different colours taking one edge from G_{bds} and G_{bdd}.

Let us illustrate how to form a Latin square with all the distinct perfect matchings of a $K_{n,n}$. Consider $G = K_{4,4}$ with the vertex set $L = \{a, b, c, d\}$ and $R = \{1, 2, 3, 4\}$. We represent four of its distinct perfect matchings by a 4×4 Latin square where each row is labeled by a distinct vertex in L and each column represents a distinct perfect matching. Therefore, in our representation for an entry (r, c, s), s is the vertex in R which is the mate of the vertex in L associated with its r^{th} row in the c^{th} perfect matching. The four perfect matchings need to be such that each vertex of R appears exactly once in a row and exactly once in a column of the Latin square.

Here, a set of four matchings forms a distinct Latin square, but not all the Latin squares can produce a valid path for *TOSPS-Q*. Our observation is that only when all the

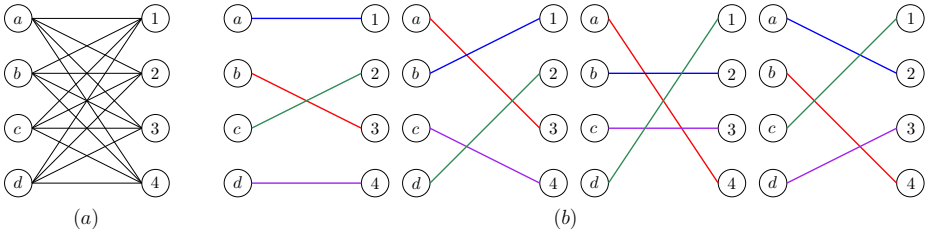


Figure 5.6: (a) $K_{4,4}$ and (b) a cover with four perfect matchings.

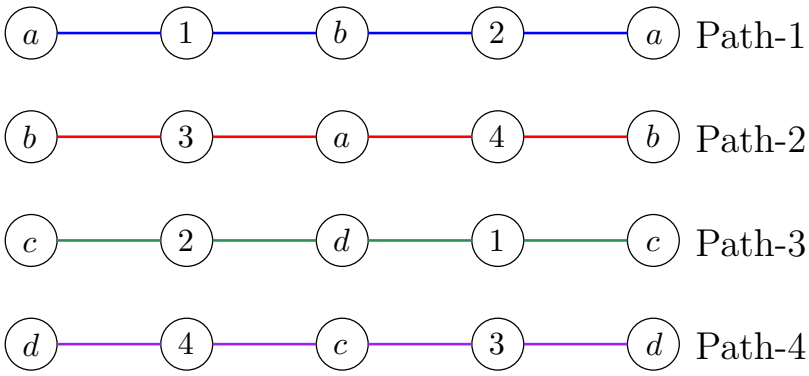


Figure 5.7: Four loops (not path) are formed while connecting the matchings obtained in Fig. 5.6(b).

elements in each row and column of a Latin square are in either lexicographic ascending or descending order, then the set of perfect matchings obtained from the Latin square constitute a valid path as shown in the Figure 5.8. Therefore, there exist at least $n-1$ valid distinct Latin squares (corresponding to $n-1$ circular shifts) each of which yields a valid path for *TOSPS-Q*. Algorithm 13 describes the steps to select $n-1$ edge disjoint paths by leveraging the set of $n-1$ matchings.

In Figure 5.5, we show three edge sets corresponding to three distinct perfect matchings for G_{icb} which is $K_{3,3}$; and the formation of the three edge-disjoint paths each of length three. These complete $s-d$ paths, each of length five, are obtained by including the two appropriate boundary edges, one each from G_{bds} and G_{bdd}. Now, the pertinent question is whether any set of distinct perfect matchings covering all the edges of G_{icb} are suitable for finding the paths in $\mathcal{P}_{n-1}$.

Case Q2: $l_p < n-1$

By incorporating this constraint, *TOSPS-Q* problem reduces to *finding the minimum number of paths of length* l_p *in* G_{icb} *covering all the edges in* E_i. In (MSB19), the authors

Algorithm 13: Path finding using matching in G_{icb}

Data: The graph $G_{icb} = K_{n-1,n-1}$, with $L = \{1, 2, \ldots (n-1)\}$ and $R = \{n, (n+1, \ldots (2n-2)\}$

Result: $\mathcal{P}_{n-1}$: a set of $(n-1)$ paths, each of length $(n-1)$

```
1  M_complete = ∅; a list of lists to store n − 1 matchings, each of size n − 1;
2  for i ← 0 to (n − 1) do
3  |   M_i = φ ;
4  |   Find a complete matching in G_icb using a Latin square with the entries in either
   |    ascending or descending order, and store it in M_i;
5  |   Append M_i to the list M_complete;
6  |   Delete all edges in M_i from the graph G_icb
7  end
8  j = 1;
9  while j ≤ (n − 1) do
10 |   P_j = φ, to store the n − 1 edge of a selected path in G_icb;
11 |   v_s = j, start vertex of a path;
12 |   for matching ← M_complete do
13 |   |   Append to P_j only the matched edge (v_s, v_t| v_s, v_t ∈ L ∪ R) in matching ;
14 |   |   v_s ← v_t, store the other end vertex for next search;
15 |   |   Delete the edge (v_s, v_t) from all the matchings in M_complete;
16 |   end
17 |   Append the path P_j to P_{n−1};
18 |   j = j + 1;
19 end
```

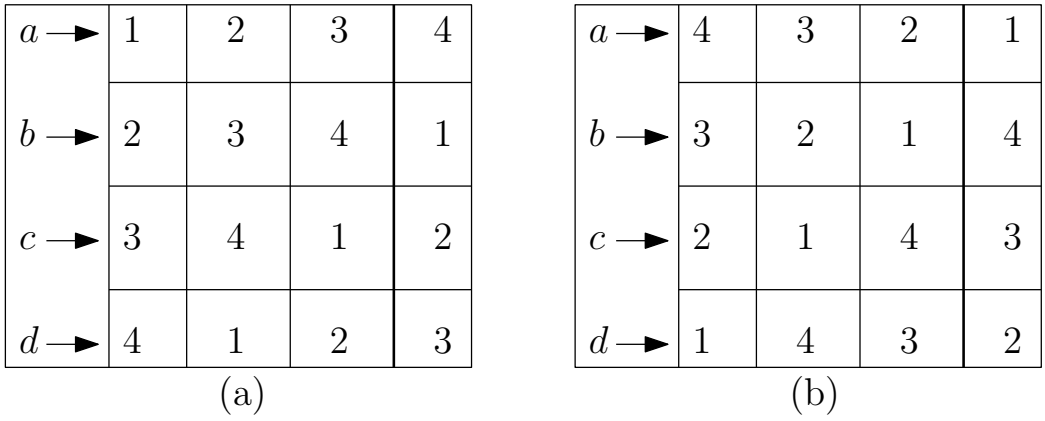


Figure 5.8: Latin square with (a) naturally increasing order, (b) naturally decreasing order.

proposed a method where an Eulerian cycle is constructed from the graph G_{icb} ($K_{n-1,n-1}$), and the paths are generated by decomposing the cycle into consecutive paths of length l_p.

For any tour in the graph $K_{n-1,n-1}$ which covers each edge exactly once, each vertex appears exactly $\frac{n-1}{2}$ times. However, such a tour may not guarantee that a sub-tour of length l_p will have no repetition of a vertex. In order to overcome this difficulty, we construct a tour by concatenating the paths selected by Algorithm 13 with the help of a Latin square (KD15). This construction guarantees that there is no repetition of any vertex for any chosen sub-tour section of length $l_p < n-1$. While constructing the tour, we concatenate all the $n-1$ paths sequentially as produced by Algorithm 13, i.e., the path $\mathcal{P}_{n-1}[0]$ in Algorithm 13 is concatenated with the path $\mathcal{P}_{n-1}[1]$ followed by $\mathcal{P}_{n-1}[2]$ and so on as in Algorithm 14.

In Algorithm 14, two consecutive paths are concatenated via two unused vertices not present in those paths in order to ensure that no vertex repeats within the length $\frac{n-1}{2}$ in the output cycle. In Lemma 5.4.1, we show that for $n \geq 6$ there always exist two unused vertices, one in L and the other in R, while augmenting these two consecutive paths. Let $\mathcal{P}_{n-1}$ be the set of $n-1$ paths, each of length $n-1$, produced by Algorithm 13. Let $V_l^i, V_l^{i+1} \in L$ be the two sets of vertices of the paths $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$ respectively, and $V_L^i = V_l^i \cup V_l^{i+1}$. Similarly, let $V_r^i, V_r^{i+1} \in R$ be the two sets of vertices of the paths $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$ respectively, and $V_R^i = V_r^i \cup V_r^{i+1}$. Let the index sequence for $L = \{1, 2, 3, \ldots, n-1\}$ and for $R = \{n, n+1, n+2, \ldots, 2n-2\}$.

Lemma 5.4.1. *For a given bipartite graph G_{icb} ($K_{n-1,n-1}$) where $n \geq 6$, and its paths*

Algorithm 14: Concatenation of paths obtained by Algorithm 13 in the graph $G_{icb}(K_{(n-1),(n-1)})$

Data: $\mathcal{P}_{n-1}$, set of $n-1$ paths generated by Algorithm 13
Result: Cycle of length $(n-1)^2+3(n-2)$ and $(n-1)^2+(n-2)$ for odd and even $n-1$, respectively

$i = 0$;
while $i \leq n-1$ **do**
  The vertices for paths $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$
  $V^i = V_l^i + V_r^i$, $V^{i+1} = V_l^{i+1} + V_r^{i+1}$
  $V_l^i, V_l^{i+1} \subset L$, $V_l^r, V_l^{r+1} \subset R$;
  Find $v_l \in L \setminus V_l^i \cup V_l^{i+1}$ and $v_r \in L \setminus V_r^i \cup V_r^{i+1}$;
  if $n-1$ *is odd* **then**
    Concatenate the path $\mathcal{P}_{n-1}[i+1]$ to $\mathcal{P}_{n-1}[i]$ via vertices v_l and v_r;
  else
    Join paths $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$ via vertices v_r;
  end
  $i = i+1$;
end

$\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$, $0 \leq i \leq n-1$, found by Algorithm 13 in consecutive iterations, there exists two vertices $v_l^i \in L \setminus V_L^i$ and $v_r^i \in R \setminus V_R^i$.

Proof. In Algorithm 13, Let the index sequence for $\mathcal{P}_{n-1}[i]$
$V_l^i = \{i, (i-1), \ldots, 2, 1, (n-1), (n-2), \ldots, (n-k)\} \subset L$;
$V_r^i = \{(n+i-1), (n+i), \ldots, (2n-2), n, \ldots, (n+l)\} \subset R$;
where $k, l \in \mathbf{N}$
and for $\mathcal{P}_{n-1}[i+1]$
$V_l^{i+1} = \{i+1, i, \ldots, 2, 1, (n-1), (n-2), \ldots, (n-k-1)\} \subset L$
$V_r^{i+1} = \{(n+i), (n+i+1), \ldots, (2n-2), n, \ldots, (n+l+1)\} \subset R$
The sets of vertices in L and R common to $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$ are
$V_l^{i,i+1} = V_l^i \cap V_l^{i+1} = \{i, (i-1), \ldots, 2, 1, (n-1), (n-2), \ldots, (n-k-1)\}$
$V_r^{i,i+1} = V_r^i \cap V_r^{i+1} = \{(n+i), (n+i+1), \ldots, (2n-2), n, \ldots, (n+l)\}$
It follows that
$V_L = V_l^{i,i+1} \cup \{i+1\} \cup \{n-k\}$

$V_R = V_r^{i,i+1} \cup \{n+i-1\} \cup \{n+l+1\}$
Since $|V_l^i| = |V_l^{i+1}| = \lfloor \frac{n-1}{2} \rfloor + 1$; $|V_r^i| = |V_r^{i+1}| = \lceil \frac{n-1}{2} \rceil$
therefore, $|V_L| = |V_l^{i,i+1}| + 2 = \lfloor \frac{n-1}{2} \rfloor + 2$;
$|V_R| = |V_r^{i,i+1}| + 2 = \lceil \frac{n-1}{2} \rceil + 1$
The number of unused vertices in $\mathcal{P}_{n-1}[i]$ and $\mathcal{P}_{n-1}[i+1]$ are, $|L| - |V_L| = (n-1) - (\lfloor \frac{n-1}{2} \rfloor + 2) = \lceil \frac{n-1}{2} \rceil - 2 > 0$
$|R| - |V_R| = (n-1) - (\lceil \frac{n-1}{2} \rceil + 1) = \lfloor \frac{n-1}{2} \rfloor - 1 > 0$. □

5.4.3 Test Optimization by Multiple Paths Selection (TOMPS-Q)

Here, we select multiple paths from source vertex s to destination vertex d in G_{spn}, and test them simultaneously. Therefore, our problem reduces to that of *finding a maximum number of vertex-disjoint paths each having one edge in* G_{icb}. This is equivalent to finding a set of $n-1$ perfect matchings each of size $n-1$ in G_{icb} ($K_{n-1,n-1}$). All the matched edges in each perfect matching can be tested simultaneously. For each perfect matching in G_{icb}, the matched edges are augmented to one edges each from G_{bds} and G_{bdd}, respectively, to get $n-1$ parallel paths, each of length three in G_{spn}. For $n-1$ edge disjoint matchings, the total testing time is equal to $n-1$. Figure 5.9 illustrates four matchings that covers all edges of the graph $G_{icb} = K_{4,4}$.

5.5 Testing of Full Rectangular Crossbars

Let $G_{spn}(V, E)$ be the graph corresponding to an $m \times n$ memristive crossbar where $m > n$, $|V| = m+n$ and $|E| = mn-1$. For the simplicity of the analysis, based on the values of m and n, the crossbars are split into two categories and the testing algorithms are designed accordingly.

Case-R1: $\lceil \frac{m-1}{2} \rceil \leq (n-1) \leq (m-1)$

$G_{icb}(V_i, E_i) = K_{m-1,n-1}$ is the intermediate bipartite graph where $|L| = m-1$, $|R| = n-1$ and $|E_i| = (m-1) * (n-1)$. In this method the complete path selection is done in two discrete steps. In the first step, $|L| - |R| = m-n$ dummy vertices is augmented to the vetex set R and also augment $(m-1)*(m-n)$ dummy edges to convert G_{icb} into the graph $G'_{icb}(V'_i, E'_i) = K_{m-1,m-1}$. Now we apply the same heuristic used for square crossbar and select $m-1$ paths each of length $m-1$ including $m-n$ dummy vertices and $(m-1)*(m-n)$ dummy edges as given in Algorithm 15.

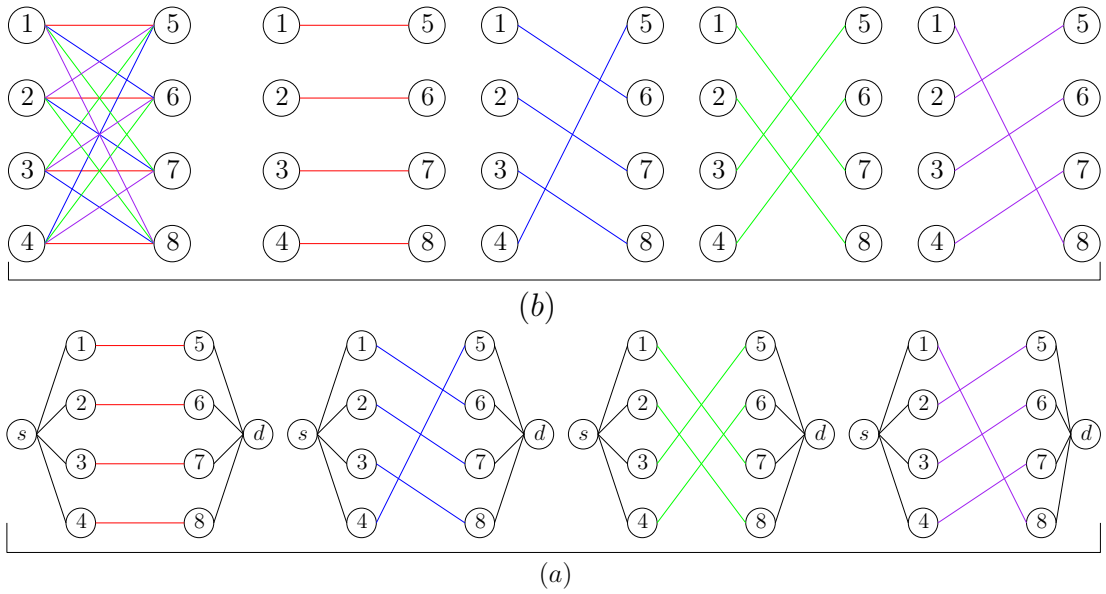


Figure 5.9: (a) $K_{4,4}$ G_{icb} and a cover with four edge-disjoint perfect-matching in G_{icb}, and (b) four complete sneak-paths in G_{snp} indicated by different colours taking all edges from G_{bds} and G_{bdd}.

In the final step, as $m-n$ dummy vertices, as well as $(m-1)*(m-n)$ dummy edges, are present in the solution paths, for each path in the selected path set all the dummy vertices (if any) along with the dummy edges (if any) are deleted. Then the disconnected path sections (created due to deletion of vertices and edges) are stitched together by utilizing real vertices and real edges not present in that path as described in Algorithm 16. After the deletion of all the dummy vertices and all the dummy edges from the selected paths the length of all the paths or a section, the path is reduced but the overall counting remains the same.

Therefore, for a rectangular $m \times n$ crossbar, Algorithms 15&16 provide $m-1$ paths of various length covering all the edges. Figure 5.10 describes the selection of five paths including dummy vertices and dummy edges as well as the deletion of dummy vertices and edges and the stitching of path sections for a 5×3 rectangular crossbar.

Now, in Algorithm 16, our argument is that after the deletion of vertices if a selected path is decomposed into k sub-paths then there must be $k-1$ real vertices not present in that path which are to be used to join these k sub-paths into a single path of length $l_p \leq (m-1)$. In Lemma 5.5.1, we ensure that there always be an unused real vertex for

Algorithm 15: Selection of $m-1$ paths including the dummy vertices and edges

Data: G_{icb} corresponding to an $m \times n$ memristive crossbar
Result: $\mathcal{P}'_{m-1}$: Set of $m-1$ paths each of length $m-1$ including dummy edges

$V_m = m-1$: Vertex set containing $m-1$ vertices;
$V_n = n-1$: Vertex set containing $n-1$ vertices;
Add $m-n$ dummy vertices to the set V_n;
Add $(m-1)*(m-n)$ dummy edges to convert G_{icb} into a complete bipartite graph $(K_{m-1,m-1})$;
Apply Algorithm 13 to find $\mathcal{P}_{m-1}$

Algorithm 16: Removal of dummy vertices and edges from the selected path and stitching of disjoint path sections.

Data: $\mathcal{P}'_{m-1}$ solution paths including dummy edges.
Result: $\mathcal{P}_{m-1}$ solution paths containing real vertices only

V_r: Set containing $(m-1)(n-1)$ real vertices;
V_d: Set containing $m-n$ dummy vertices;
for $path, P \leftarrow \mathcal{P}'_{m-1}$ **do**
    $V_p = V_p^r \cup V_p^d \; | V_p^r \subset V_r, V_p^d \subset V_d$: Set of vertices in path **for** $vertex, v \leftarrow V_p$ **do**
        if $v \in V_d$ **then**
            Delete the two edges incident on v
        else
            pass;
        end
    end
    Remove all the isolated vertices;
    $V_{left} = V_r \setminus V_p^r$;
    Stitch the disconnected path sections using $v \in V_{left}$ and corresponding edges;
    Append the path to $\mathcal{P}_{m-1}$;
end
Report $\mathcal{P}_{m-1}$ containing real vertices and real edges only;

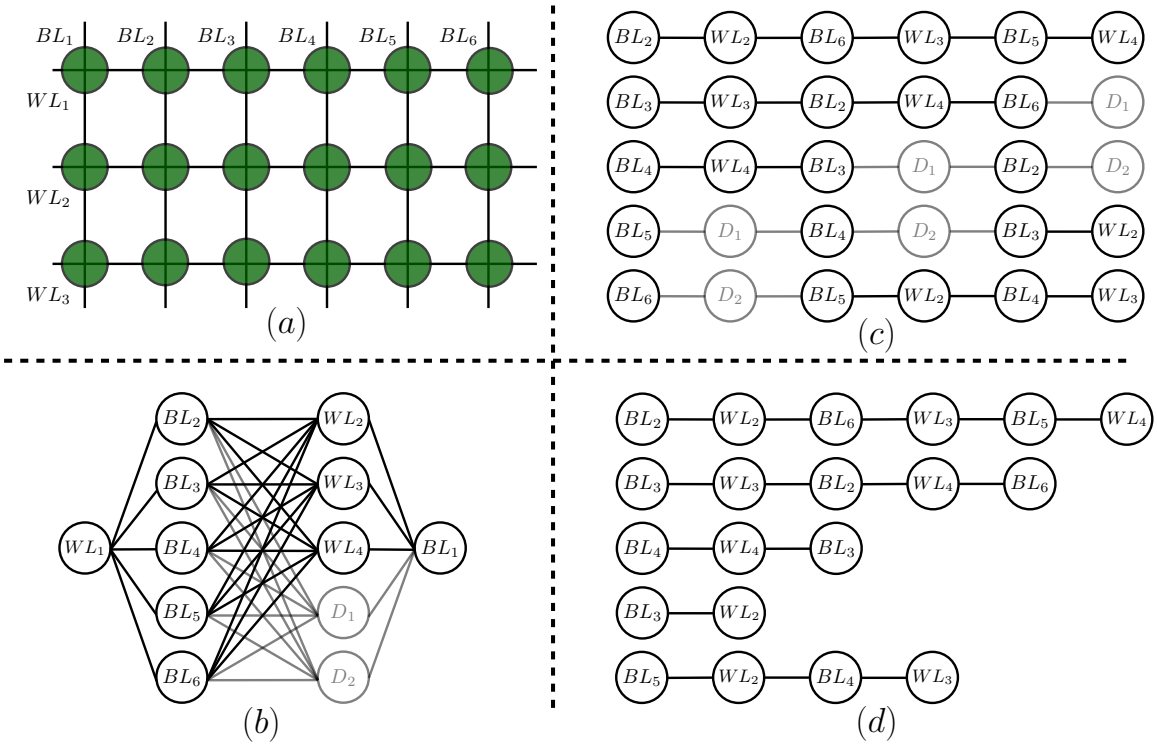


Figure 5.10: (a) An 5×3 crossbar (b) corresponding G_{spn} by adding two dummy vertices and 10 dummy edges (c) five selected paths including dummy vertices and edges (d) five paths of various length containing real edges only.

every single or multiple consecutive dummy vertices.

Lemma 5.5.1. *For each path in the selected paths set, if k dummy vertices are present on that path then there exist k real vertices not present on that path.*

Proof. The paths are selected from the graph $K_{m-1,m-1}$ by Algorithm 15. In each selected path there are m vertices ($m-1$ edges).

Let $V_p = V_p^L \cup V_p^R = |m|$ be the set of vertices present in a selected path p such that $V_p^L \in L, V_p^R \in R$ where $|V_p^L| = \lfloor \frac{m-1}{2} \rfloor + 1$ and $V_p^R| = \lceil \frac{m-1}{2} \rceil$, respectively. Let also assume that $V_p^R = V_p^R(real) \cup V_p^R(dummy)$ where $V_p^R(real)$ and $V_p^R(dummy)$ be the set of real and dummy vertices, respectively. If $|V_p^R(dummy)| = k$ $(0 \leq k < \lfloor \frac{m-1}{2} \rfloor)$ dummy vertices are used in the selected path p then number of real vertices in p belongs to R must be

$|V_p^R(real)| = \lceil \frac{m-1}{2} \rceil - k$. Therefore there are at least $\lceil \frac{m-1}{2} \rceil - (\lceil \frac{m-1}{2} \rceil - k) = k$ unused real vertices in p. □

Case-RII: $(n-1) < \lceil \frac{m-1}{2} \rceil$

Here, the number of dummy vertices required in order to convert $K_{m-1,n-1}$ to $K_{m-1,m-1}$ being high, Lemma 5.5.1 does hold true. For testing these type of crossbars, each time we select $n-1$ vertices (un-selected) out of $m-1$ vertices from set L and all the $n-1$ vertices from set R to construct an $K_{n-1,n-1}$ and apply the Algorithm 13 to find $n-1$ paths each of length $n-1$. This process is repetitively done until all vertices in set L are covered. Therefore, in order to test the complete graph $K_{m-1,n-1}$, total number of selected path is $\lceil \frac{m-1}{n-1} \rceil \times (n-1)$. While comparing the number of selected paths for testing a rectangular crossbar we find that in this case excess number of paths is $(m-1)-(\lceil \frac{m-1}{n-1} \rceil \times (n-1)) = n-2$. Figure 5.11 demonstrates path selection in the graph G_{icb}.

5.5.1 Test Optimization by Multiple Paths Selection (TOMPS-R)

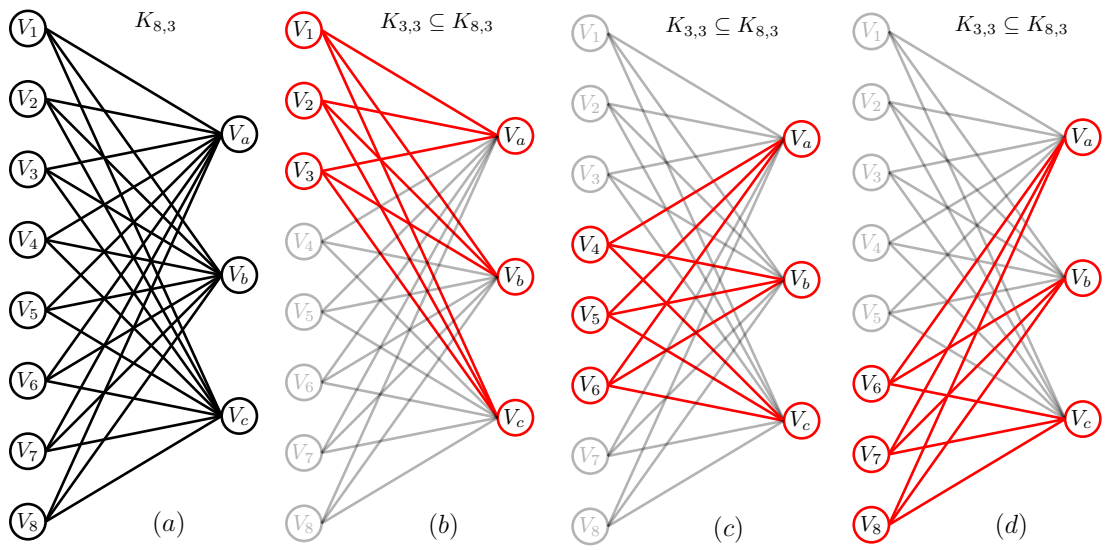


Figure 5.11: Illustrating path selection method on G_{icb}: (a) $(L \cup R, E) = K_{8,3}$ where $L = \{v_1, v_2, ..., v_8\}$, $R = \{v_a, v_b, v_c\}$, by decomposing into the three subgraphs (each being a $K_{3,3}$) namely, (b) $G_1 = (L_1 \cup R, E_1)$, (c) $G_2 = (L_2 \cup R, E_2)$ and (d) $G_3 = (L_3 \cup R, E_3)$, where $L_1 = \{v_1, v_2, v_3\}, L_2 = \{v_4, v_5, v_6\}$, $L_3 = \{v_6, v_7, v_8\}$ and $L = L_1 \cup L_2 \cup L_3$.

Testing of multiple parallel paths together in an $m \times n$ crossbar is done in the same

way as in for square one in Section 5.4. Since here $m > n$,each time maximum $n-1$ number of vertex disjoint parallel paths of length three is activated from s to d in G_{spn}. Therefore, from the G_{icb}, each time we construct an $K_{m_i,n-1}$ such that $V_m^L(i) \subset V_{m-1}^L$ where $V_m^L(i)$ is the set of vertices selected for i^{th} iteration and $|V_m^L(i)| = n-1$. For each $K_{m_i,n-1}$, there are $n-1$ perfect matching which collectively cover all the edges in $K_{m_i,n-1}$. Therefore testing time is equal to $n-1$. Now, as there are $\lceil \frac{m-1}{n-1} \rceil$ subgraphs to cover all the edges, the total testing time is proportional to $\lceil \frac{m-1}{n-1} \rceil \cdot (n-1)$.

5.6 Evaluation of the proposed methods

Table 5.1: Time to test different types of faults, depending on l_p for a complete square crossbar.
[l_p = Path Length, depends on memristor parameters, fault types, applied voltage and current sense amplifier]

$\alpha = (n-1)$, $\gamma = \lceil \frac{(n-1)^2+3(n-2)}{l_p} \rceil$, $\delta = \lceil \frac{(n-1)^2+(n-2)}{l_p} \rceil$				
Fault	SPS			MPS
	$l_p \geq n-1$	$l_p < n-1$		
		n is odd	n is even	
SA0	$\alpha * (t_{w1} + t_{r1})$	$\gamma * (t_{w1} + t_{r1})$	$\delta * (t_{w1} + t_{r1})$	$\alpha * (t_{w1} + t_{r1})$
SA1	$\alpha * (t_{w0} + t_{r0})$	$\gamma * (t_{w0} + t_{r0})$	$\delta * (t_{w0} + t_{r0})$	$\alpha * (t_{w0} + t_{r0})$
D0	$\alpha * (2t_{w0} + t_{w1} + t_{r1})$	$\gamma * (2t_{w0} + t_{w1} + t_{r1})$	$\delta * (2t_{w0} + t_{w1} + t_{r1})$	$\alpha * (2t_{w0} + t_{w1} + t_{r1})$
D1	$\alpha * (2t_{w1} + t_{w0} + t_{r0})$	$\gamma * (2t_{w1} + t_{w0} + t_{r0})$	$\delta * (2t_{w1} + t_{w0} + t_{r0})$	$\alpha * (2t_{w1} + t_{w0} + t_{r0})$
SW0	$\alpha * (t_{w1} + t_{w0} + t_{r0})$	$\gamma * (t_{w1} + t_{w0} + t_{r0})$	$\delta * (t_{w1} + t_{w0} + t_{r0})$	$\alpha * (t_{w1} + t_{w0} + t_{r0})$
SW1	$\alpha * (t_{w0} + t_{w1} + t_{r1})$	$\gamma * (t_{w0} + t_{w1} + t_{r1})$	$\delta * (t_{w0} + t_{w1} + t_{r1})$	$\alpha * (t_{w0} + t_{w1} + t_{r1})$

Reading *logic*-0 and *logic*-1 from the memristors and writing *logic*-0 and *logic*-1 into the memristors are two basic operations for memristor testing. We deploy these four standard memristor-access operations as mentioned below (BA00):

- $\{w0, w1\} \Rightarrow$ Writing *logic*-0 and *logic*-1 into a memristor, respectively.
- $\{r0, r1\} \Rightarrow$ Reading *logic*-0 and *logic*-1 from a memristor, respectively.

We apply appropriate combinations of four access-modes sequentially in order to test all faults considered in the fault model. For each selection, we are only considering one type of fault and for a particular selected path, fault detection is performed for each type of fault in the fault model one by one. Sensitization of a particular sneak-path allows concurrent write of *logic*-0 and *logic*-1 in all the memristors present on the selected path. Consequently,

this technique leads to a reduction of test-time as demonstrated below.

***Stuck-at-*0 ($SA0$): A $SA0$ fault is detected by applying the sequence $\{w1, r1\}$. In the first step, *logic*-1 is written concurrently into all the memristors of the selected path. Then read the path by applying a predetermined voltage source across the path with expected *logic*-1 in all the memristors in the path. $SA0$ fault is detected by sensing the difference between expected current and actual current in the path. When $l_p \geq n-1$ in an $n \times n$ crossbar, $n-1$ tests are sufficient for writing $\{w1\}$ and for reading $\{r1\}$ whereas for $l_p \leq n$, test time will be $\lceil \frac{(n-1)(n-1)}{l_p} \rceil$ and $\lceil \frac{(n-1)(n-2)}{l_p} \rceil + \lceil \frac{2(n-1)}{l_p} \rceil$ for even and odd n, respectively. As all the memristors in the path are set to *logic*-1, the equivalence memristance of the path is very low therefore current value is high. So single path sensitization is preferable for this this fault testing.

***Stuck-at-*1 ($SA1$): The sequence $\{w0, r0\}$ detects a $SA1$ fault. Here, *logic*-0 is written into all the memristors in the path, and then a read voltage is applied with expected *logic*-0 in all the memristor in the path. $SA1$ fault is sensed from the difference in the expected, and actual current value. In this case, as all the memristors of the selected path are set to *logic*-0, the equivalence memristance of the path is very high and subsequently, current value is low. Therefore multiple parallel path sensitization is more effective for detecting these faults. A total of $(n-1)$ tests cover all such faults in the entire $n \times n$ crossbar.

***Deep-*0 ($D0$): The sequence $\{w0, w0, w1, r1\}$ is applied to test deep-zero faults. In order to take the memristor into *Deep-0* state two consecutive $w0$ pulses are applied. The next pulses are the same as $SA0$ detection. Due to the application of three consecutive write pulses, write-time is three-time more than that of read-time.

***Deep-*1 ($D1$): For testing $D1$, the following sequence $\{w1, w1, w0, r0\}$ is applied. In order to take the memristor into *Deep-1* state two consecutive $w1$ pulses are applied. The next pulses are same as $SA1$ detection. Here also the write-time is three time more than read-time.

***Slow-write-*0 ($SW0$): The following sequence is applied to test a $SW0$ fault: $\{w1, w0, r0\}$. In this case, write-time is two times larger than read-time.

***Slow-write-*1 ($SW1$): Applied sequence is $\{w0, w1, r1\}$ with write-time two times larger than read-time.

Due to the concurrent read and write operation on all the memristors in the selected path, test time is proportional to the number of selected paths in the crossbar. Let, the write time and read time for *logic*-0 and logic-1 of the selected path (irrespective of its length) is defined as below.

Table 5.2: Time to test for various types of faults depending on l_p in a rectangular crossbar.
[l_p = Path Length, depends on memristor parameters, fault types, applied voltage and current sense amplifier]

$\alpha = (n-1)$, $\sigma = (m-1)$, $\phi = \lceil \frac{(m-1)}{n-1} \rceil$			
Fault	SPS		MPS
	$n-1 \geq \frac{m-1}{2}$	$n-1 < \frac{m-1}{2}$	
SA0	$\sigma * (t_{w1} + t_{r1})$	$\phi * \alpha * (t_{w1} + t_{r1})$	$\phi * \alpha * (t_{w1} + t_{r1})$
SA1	$\sigma * (t_{w0} + t_{r0})$	$\phi * \alpha * (t_{w0} + t_{r0})$	$\phi * \alpha * (t_{w0} + t_{r0})$
D0	$\sigma * (2t_{w0} + t_{w1} + t_{r1})$	$\phi * \alpha * (2t_{w0} + t_{w1} + t_{r1})$	$\phi * \alpha * (2t_{w0} + t_{w1} + t_{r1})$
D1	$\sigma * (2t_{w1} + t_{w0} + t_{r0})$	$\phi * \alpha * (2t_{w1} + t_{w0} + t_{r0})$	$\phi * \alpha * (2t_{w1} + t_{w0} + t_{r0})$
SW0	$\sigma * (t_{w1} + t_{w0} + t_{r0})$	$\phi * \alpha * (t_{w1} + t_{w0} + t_{r0})$	$\phi * \alpha * (t_{w1} + t_{w0} + t_{r0})$
SW1	$\sigma * (t_{w0} + t_{w1} + t_{r1})$	$\phi * \alpha * (t_{w0} + t_{w1} + t_{r1})$	$\phi * \alpha * (t_{w0} + t_{w1} + t_{r1})$

- t_{w0}, t_{w1}: Time required for writing *logic*-0 and *logic*-1, respectively, into all the memristors of a selected path.
- t_{r0}, t_{r1}: Time required for reading *logic*-0 and *logic*-1, respectively, from all the memristors of a selected path.

While calculating test time for a particular fault, we multiply the number of paths with the number of reading time and write time required to detect the fault and add them together to get the total testing time for that fault. In Table 5.1 and Table 5.2, total testing time is calculated for different approach for a square complete crossbar of size $n \times n$ and a rectangular crossbar of size $m \times n$ where $m > n$, respectively.

In order to compare our results with the previously published works, we assume $t_{w0} = t_{w1} = t_{r0} = t_{r1} = 1$ and represent the test time in terms of selected paths only. We also assume that there is no constraint on the length (l_p) of the selected path and hence for comparison we have taken the results for the square complete crossbar with path length $l_p \geq (n-1)$. We find that in all earlier work (BA00), (HH11), (KRKS13a), (SZZM18a) test time varies quadratically with n. On the other hand, in the proposed scheme, the total test-time (including both read- and write-time) grows linearly with n as shown in Table 5.3.

In order to have a visual impression, in Figure 5.12, test time by our proposed method for detecting $SA0$ fault for different sizes of the memristive crossbar is compared with the proposed works in (BA00) and (KRKS13b; KRKS13a), respectively. The difference in

Table 5.3: Comparison of test time between *TOSPS* and earlier methods; improvement over (KRKS13a).
[NT: Faults not covered by the test technique.]

Fault	March Algo. (BA00)	(HH11)	March MoM (KRKS13a)	TOSPS	% Improvement $\frac{(March_MoM)-(TOSPS)}{March-MoM} \times 100$	
					$8 \times 8 crossbar$	$64 \times 64 crossbar$
$SA0$	$2n^2$	$2n^2$	n^2+n	$2(n-1)$	80.55	96.97
$SA1$	$2n^2$	$2n^2$	$n^2+\frac{n^2}{15}$	$2(n-1)$	80.55	97.11
$D0$	NT	NT	$3n^2+\frac{n^2}{5}$	$4(n-1)$	86.32	98.03
$D1$	NT	NT	$3n^2+\frac{n^2}{5}$	$4(n-1)$	86.32	98.03
$SW0$	NT	$4n^2$	$2n^2+\frac{n^2}{5}$	$3(n-1)$	85.08	97.90
$SW0$	NT	$4n^2$	$2n^2+\frac{n^2}{5}$	$3(n-1)$	85.08	97.90

testing time being extremely large it is inconvenient to plot on a linear scale. As an example, difference in test time for the crossbar of size 32×32 between the work in (KRKS13a) and our proposed method is $1056-62 = 994$. For the sake of compactness and better comprehension, the test time plot given in Figure 18 is in log scale.

5.6.1 Simulation results

Simulation is performed in LTspice XVII on a 4-core 3GHz Intel Xeon processor with 32GB RAM in UBUNTU 18.04 LTS Operating System using the memristor model as given in (KFKW13) with the following memristor parameters: R_{off} = 200kΩ, R_{on} = 100Ω, R_{init} = 120kΩ, D = 10N, p = 2, I_{th} = 0.120μA (minimum change in output current to detect a fault). The parameters used in the plot denote the following currents through sensitized sneak-path.

- $IS_k(m,n)$: Read current value for single path selection in $m \times m$ crossbar in presence of n SAk faults where $m, n \in \mathbb{N}$ and $k \in \{0,1\}$.
- $IM_k(m,n)$: Read current value for multiple paths selection in $m \times m$ crossbar in presence of n SAk faulty paths where $m, n \in \mathbb{N}$ and $k \in \{0,1\}$.

SA0 Detection: In order to test $SA0$ faults, memristors are sensitized with *logic*-1. Therefore, for single path selection, the difference in overall impedance between the faulty and fault free path is very large and so is it in the values of the peak currents. The read current $IS_0(8,0)$ of $5.549mA$ and $IS_0(16,0)$ of $3.123mA$ is 100× greater than $IS_0(8,1)$ of

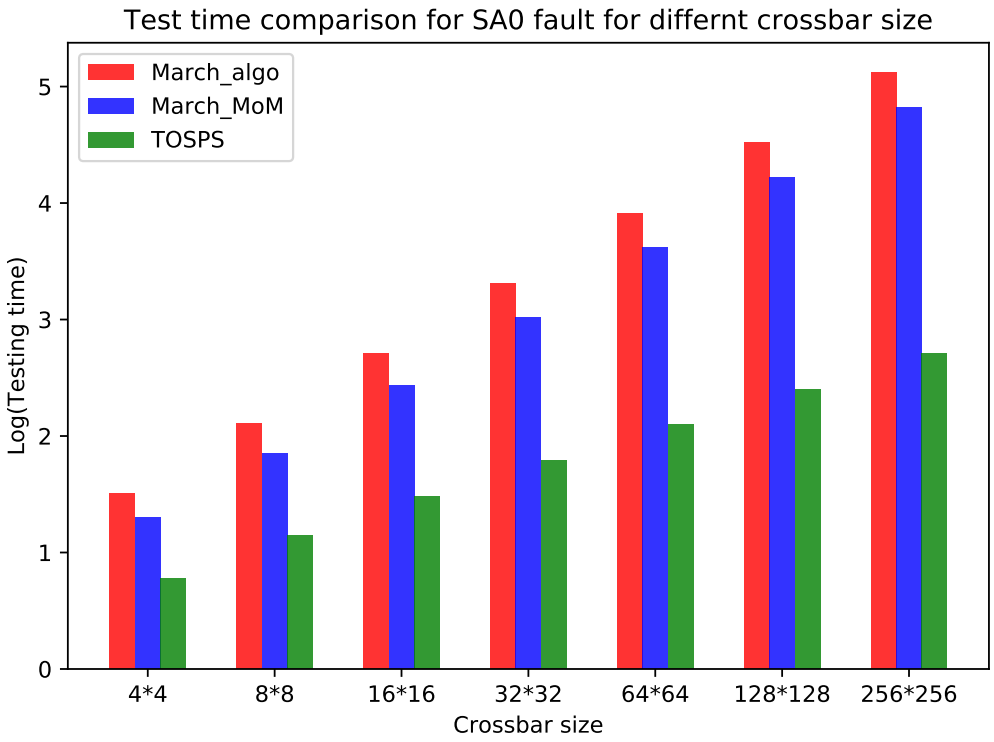


Figure 5.12: Comparison of time to test $SA0$ faults for different crossbar sizes.

$0.0496mA$) and $IS_0(16,1)$ of $0.0492mA$ for full 8×8 and 16×16, respectively. So, the current difference is easily detectable.

Figure 5.17 depicts simulation results for identifying the number of $SA0$ faults present in single path selection mode, or the number of faulty paths present in multiple paths selection modes for full 8×8 and 16×16 crossbar, respectively.

The difference in the peak currents between all pairs of $SA0$ faults for single path selection mode is shown Figure 3 in Appendix B where the minimum value is well above the I_{th}. Hence, the number of faults present in a selected path is precisely identifiable.

SA1 Detection: Figure 5.22 describes the detection of $SA1$ faults present in 8×8 and 16×16 crossbar both in single as well as multiple selection mode. While detecting $SA1$ faults, memristors are sensitized with *logic*-0. So the initial impedance is very high. As the number of faults increases the overall impedance decreases and hence the current increases. The simulation results suggest that the number of faults present in a selected path

or the number of faulty paths for multiple selections is easily identifiable as the difference in reading current for all the cases are well above the threshold current value I_{th} as shown in Figure 5.23 and Figure 5.24.

5.7 Concluding Remarks

We have addressed the problem of fault detection in a full $2D$ square and rectangular memristor-array as well as in an incomplete memristor-array with irregular structure. Various fault models such as multiple stuck-at-0/1, deep-0/1, and slow-write-0/1 faults are considered. A formulation based on a graph-theoretic representation of the sneak-path network for a given memristor-array is presented, and test generation is then abstracted as an optimal path-covering problem subject to certain constraints on path length and differentiable current sensing. For selecting paths in a square and rectangular shaped full crossbar, the proposed method relies on finding matching in a bipartite graph. All possible matchings being not eligible for path selection, a Latin square based method is next proposed for determining optimal test paths. Although for multiple memristor selection (i.e. multiple test point insertions) fewer paths may suffice, our method provides the optimum number for a single memristor selection (single test point insertion). Simulation results are presented to demonstrate the efficacy of the method. During simulation, the resistance of wires (i.e., WLs and BLs) are ignored as the differentiable current is being measured. In future, one may study test optimization for multiple memristor selection as well as how it can be used for the localization of memristor faults. Investigation on fault analysis in multi-valued memristor logic is also left as an open problem.

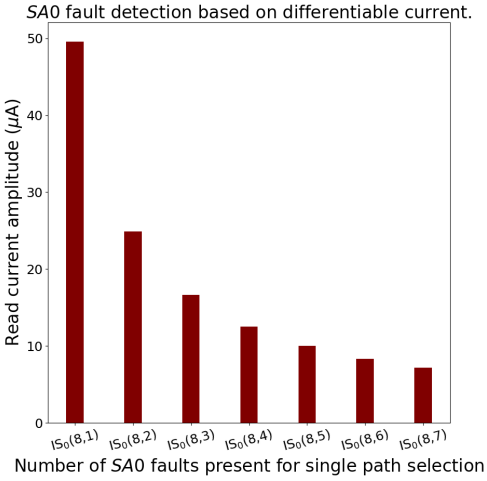


Figure 5.13: Single path for 8*8

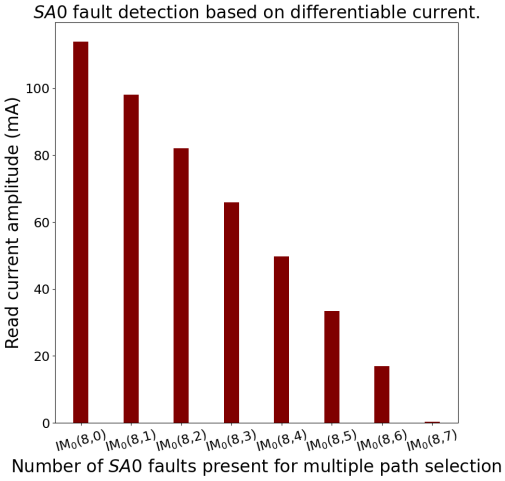


Figure 5.14: Multiple paths for 8*8

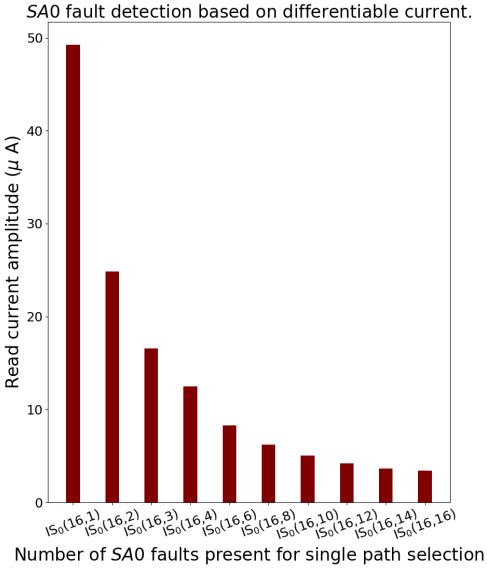


Figure 5.15: Single path for 16*16

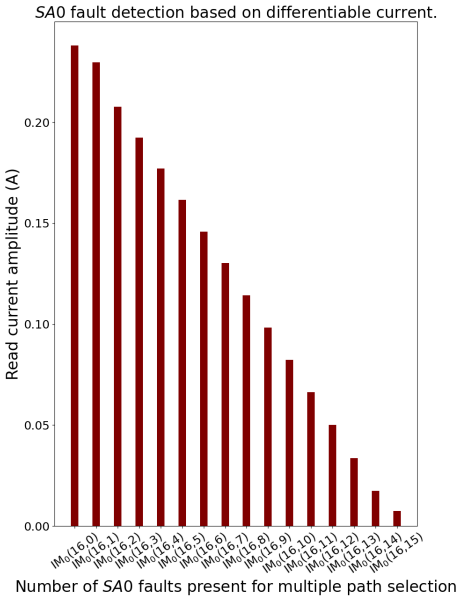


Figure 5.16: Multiple paths for 16*16

Figure 5.17: Read current in the presence of different number of $SA0$ faults for single and multiple paths selection in full 8×8 and 16×16 crossbar; $IS_0(8,0) = 5.549mA$ and $IS_0(16,0) = 3.123mA$.

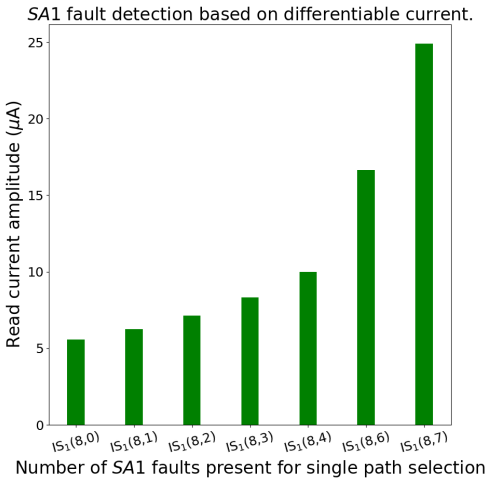


Figure 5.18: Single path for 8*8

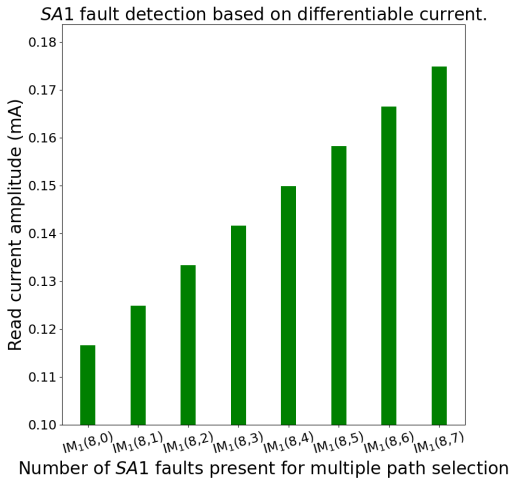


Figure 5.19: Multiple paths for 8*8

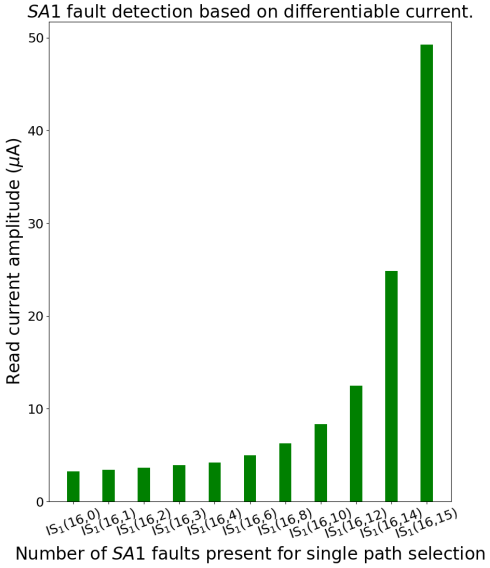


Figure 5.20: Single path for 16*16

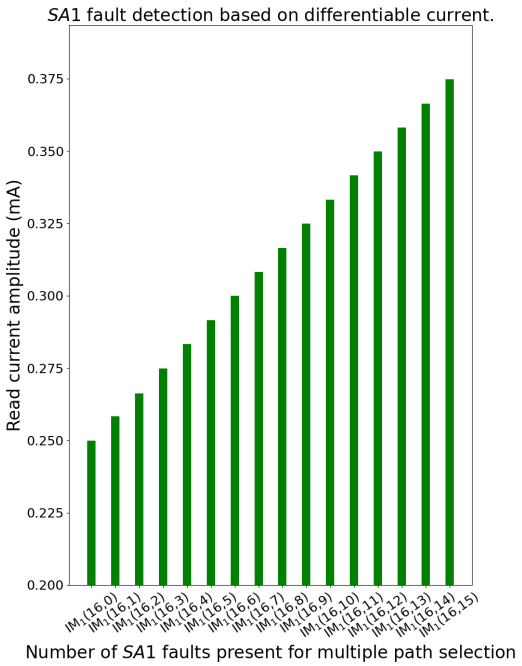


Figure 5.21: Multiple paths for 16*16

Figure 5.22: Read current in the presence of different number of $SA1$ faults for single and multiple paths selection in full 8×8 and 16×16 crossbar.

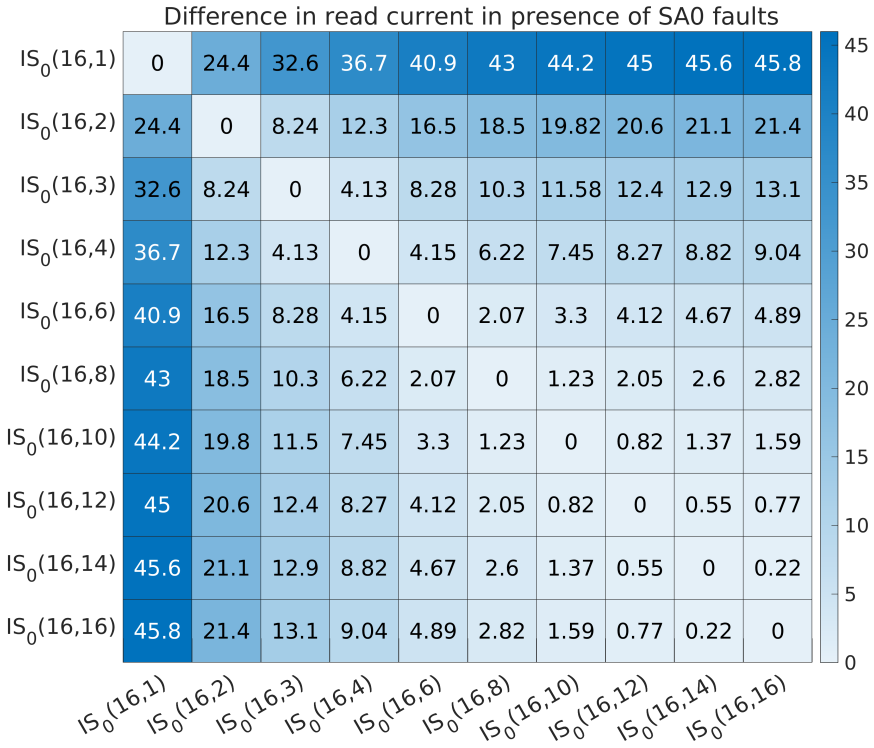


Figure 5.23: Difference in read current for single path selection in 16×16 crossbar in presence of $SA0$ faults.

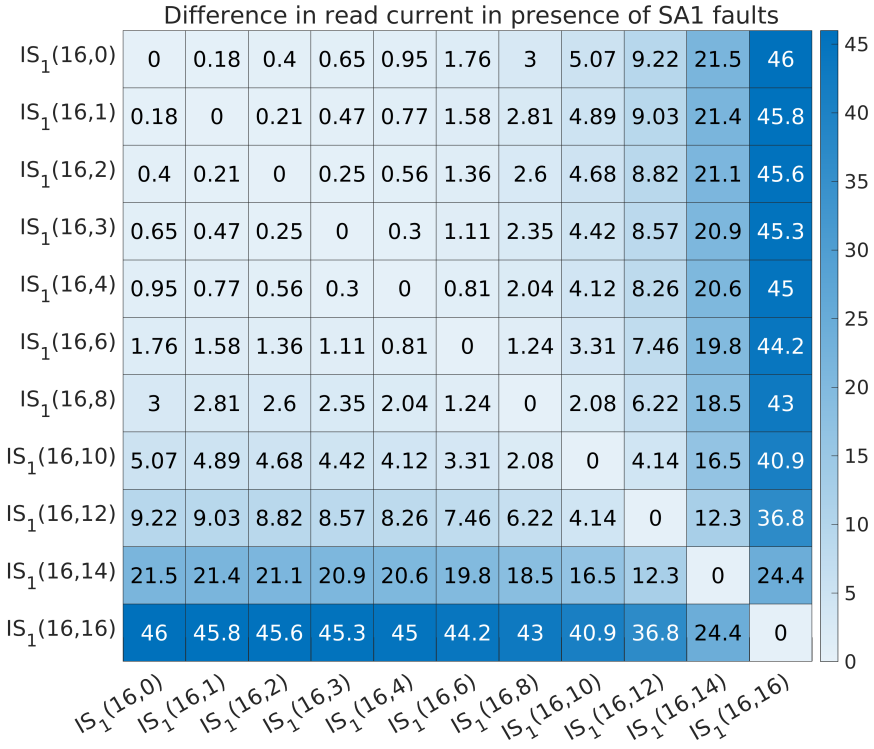


Figure 5.24: Difference in read current for single path selection in 16×16 crossbar in presence of $SA1$ faults.

CHAPTER

SIX

TEST OPTIMIZATION IN INCOMPLETE MEMRISTIVE CROSSBARS

Contents

6.1 Introduction

In Chapter 5, we have discussed testing methods for square or a rectangular full memristive crossbar i.e., there is a memristor element at each (WL, BL) cross point. However, memristive crossbar-based hardware designs have been proposed where a substantial number of memristors are missing in the crossbar, and the location(s) of these missing memristor(s) are distributed randomly in the crossbar.

We observe that the graph-based path-covering techniques proposed in Chapter 5 reduces testing time efficiently for square or a rectangular full crossbar. But, this techniques may not always be applicable for crossbar with irregular structure or for an incomplete crossbar. In order to cover all types of crossbar (including incomplete), in this chapter we have proposed an Mixed Integer Linear Program (MILP) formulation for optimal path covering that can uniformly handle both full and incomplete crossbars.

6.2 Motivation

An incomplete crossbar of size 7×4 is shown in Figure 6(a). Figure 6(b) is the graph G_{spn} representation of the crossbar as described in Chapter 5.3.2. The graph G_{spn} can be decomposed into three sub graph as shown in Figure 6.2. The intermediate bipartite graph G_{icb} corresponding to such incomplete crossbar is incomplete bipartite as shown in Figure 6.2(b). Note that the methods for path-selection discussed in Chapter 5 utilize the graph theoretic properties of complete bipartite graph, and hence cannot be readily used for incomplete bipartite graphs. Unfortunately, no efficient method is yet known for testing such a general-purpose crossbar architecture. Therefore, in this Chapter, we propose a Mixed Integer Linear Program (*MILP*) to solve the general version of the problem. Although *ILP* takes longer than our method presented in this section to select the paths for sneak path testing, it is applicable for all types of crossbar architectures.

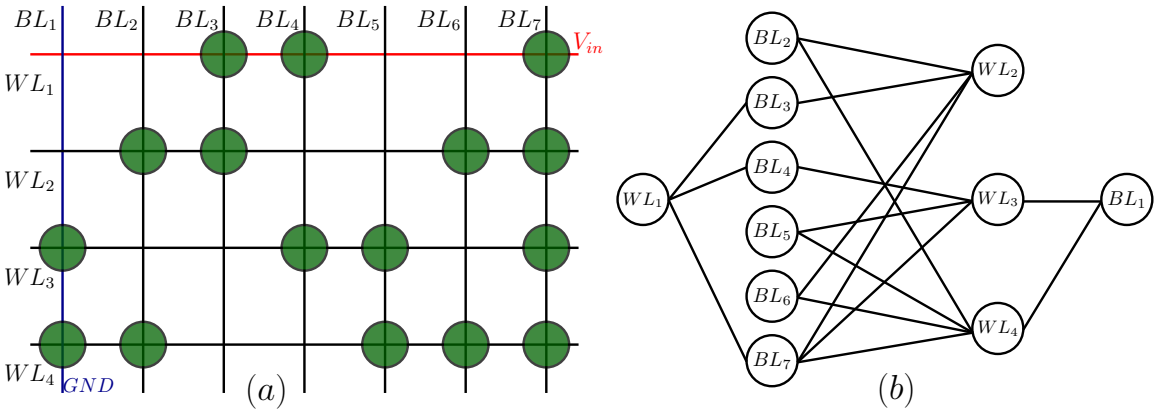


Figure 6.1: (a) An incomplete crossbar of dimension 7×4 with 16 memristor (b) the corresponding graph G_{spn} when input voltage is applied between WL_1 and BL_1.

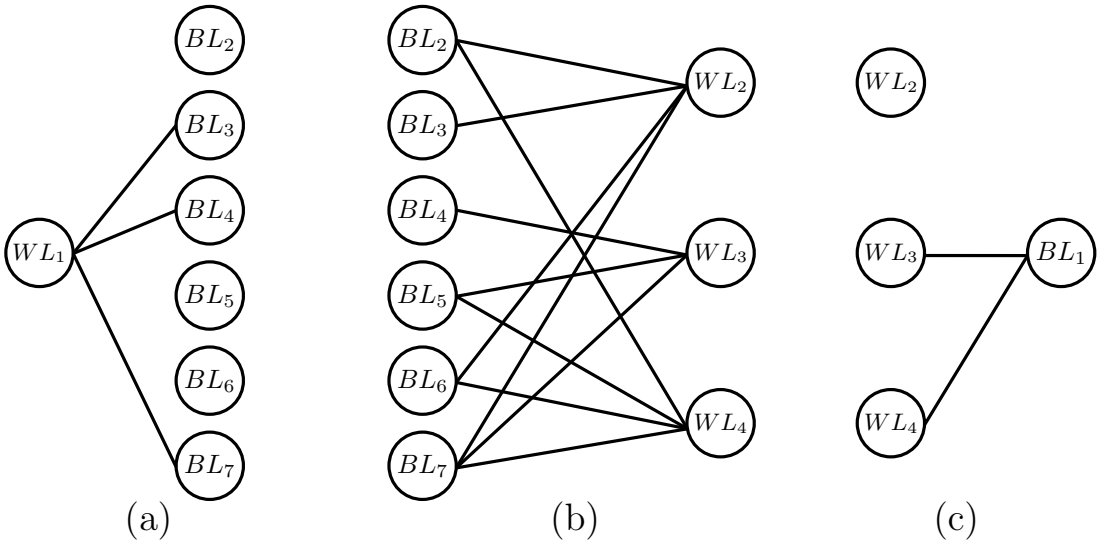


Figure 6.2: Decomposition of $G_{spn}(V, E)$ in Fig. 6.1(b): (a) G_{bds} (b) G_{icb} (c) G_{bdd}.

6.3 ILP formulation

Finding the minimum number of paths between two given vertices in an un-directed connected graph in order to cover all the edges in the graph is not possible in polynomial time as it necessitates of finding longest path between the given two vertices which is NP-hard problem. Here, in this chapter, we develop a mixed Integer Linear Program (ILP) to select minimum number of paths between two given vertices of a graph G_{spn} to cover all the edges of the graph where G_{spn} is the graph corresponds to any given crossbar architecture.

Here in Figure 6.3(a) , we have taken an undirected connected graph $G_{spn} = (V, E)$ with vertex s and d as the source and destination vertex of the graph, respectively. Now, the goal is to find a minimum number of paths from a source vertex s to a destination vertex d covering all the edges of $G_{spn} = (V, E)$. All the neighbours vertices of vertex s and d are grouped into vertex sets L and R, respectively s.t. $V = \{s\} \cup L \cup R \cup \{d\}$. We define another subgraph $G_{icb} = (V_i, E_i)$ such that $V_i = L \cup R$ and $E_i = \sum_{\forall j \in L,\ \forall k \in R} e_{jk}$.

6.3.1 Path construction

In order to select a path of length l_p in a graph $G_{spn}(V, E)$, $l_p - 1$ vertices need to be selected. If an intermediate (other than vertex s and d) vertex of degree k is selected, exactly two

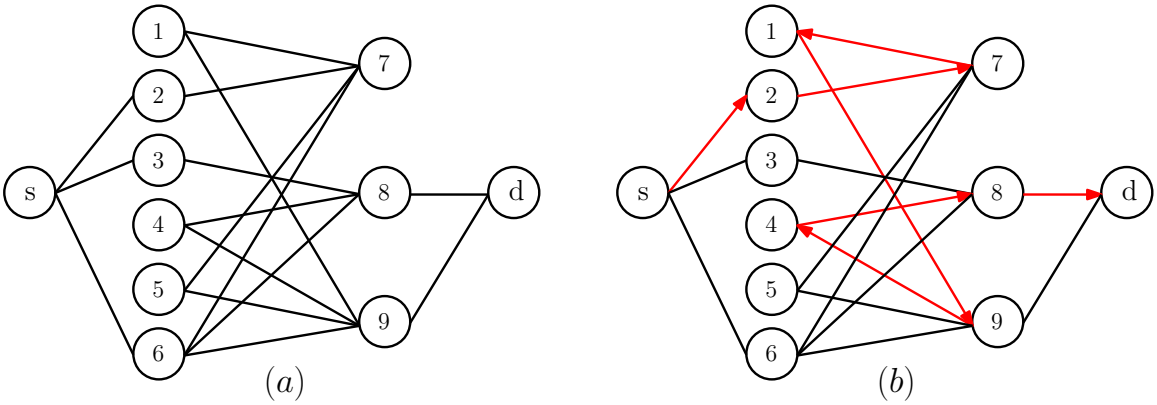


Figure 6.3: In the path construction model, (a) The graph $G(V, E)$ containing vertices s, d and vertex set L and R. (b) Construction of a path from s to d in G having maximum length.

edges incident on that vertex is selected. Since the graph is undirected, the total possibility of selection is $\binom{k}{2}$. Instead of modeling these directions directly, we model how the path passes through the surrounding vertices. Suppose all edges can be covered by no more than n_p paths in the test set, where n_p is a given constant. We assign binary (0-1) variables e_{ij}^l and v_i^l to the edges and vertices of the graph, respectively. The variable e_{ij}^l=1 if the edge $e(i, j) \in E$ is present in the l^{th} selected path and e_{ij}^l=0 otherwise. Similarly, v_i^l=1 only if the vertex $v_i \in V$ is present in the l^{th} selected path, otherwise set to 0.

In order to construct the l^{th} path from vertex s to d, exactly one edge must be selected from the edges incident in the source vertex s and destination vertex d. These constraints are written as,

$$\begin{aligned} \sum_{\forall i \in L} e_{si}^l &= 1; \\ \sum_{\forall i \in R} e_{id}^l &= 1; \end{aligned} \tag{6.1}$$

For the intermediate vertices i.e. $\forall v_i \in L \cup R$ if a vertex is selected in the l^{th} path, then exactly two edges incident on that vertex must be selected. These constraints are written as

$$\begin{aligned} \sum_{\forall i \in L,\ \forall j \in R \cup \{s\}} e_{ij}^{l} &= 2 * v_i^l; \\ \sum_{\forall i \in R,\ \forall j \in L \cup \{d\}} e_{ij}^{l} &= 2 * v_i^l; \end{aligned} \tag{6.2}$$

This constraint is formulated in order to ensure the coverage of all the edges of the graph $G(V, E)$ at least once. If the solution provided by the ILP contains n_p paths, then each edge of $G(V, E)$ lies on any of the n_p paths at least once. These constraints are formulated as

$$\sum_{l=1}^{n_p} e_{ij}^{l} \geq 1, \quad \forall\ e_{ij} \in E. \tag{6.3}$$

Equations 6.1-6.3 guarantee the construction of a path from the source vertex s to destination vertex d in a graph G_{spn}, and coverage of all the edges. But for a selected path, there might arise a close loop of even length (hereafter refer as path-loop) in the subgraph G_{icb}. For example, the above constraint does not prevent the disjoint loop as shown in Figure 6.4 from appearing in the selected path. All the edges and vertices on the loop meet the constraints 6.1 and 6.2, and this loop gives a false counting of edge coverage in path selection.

6.3.2 Disjoint path-loop removal

The constraints given by Equations 6.1-6.3 do not prevent the path-loop to arise in the solution path. As the loop is not connected to the source vertex s and destination vertex d, during testing of the path, the current does not flow through the memristor corresponding to the edges of the loop. Hence, it provides a false edge count during path selection.

To prevent the occurrence of the loop in the selected path, we define a current flow integer variable f_{ij} for each edge variable which forces the current to flow through each selected edge in the path. It must be noted that the current will flow only through the edge if the edge is present in the selected path, i.e. $f_{ij} \neq 0$ only when $e_{ij} = 1$, else $f_{ij} = 0$. These constraints are written as

$$\begin{aligned} f_{ij}^{l} &\leq M * e_{ij}^{l}; \\ f_{ij}^{l} &\geq -M * e_{ij}^{l}; \end{aligned} \tag{6.4}$$

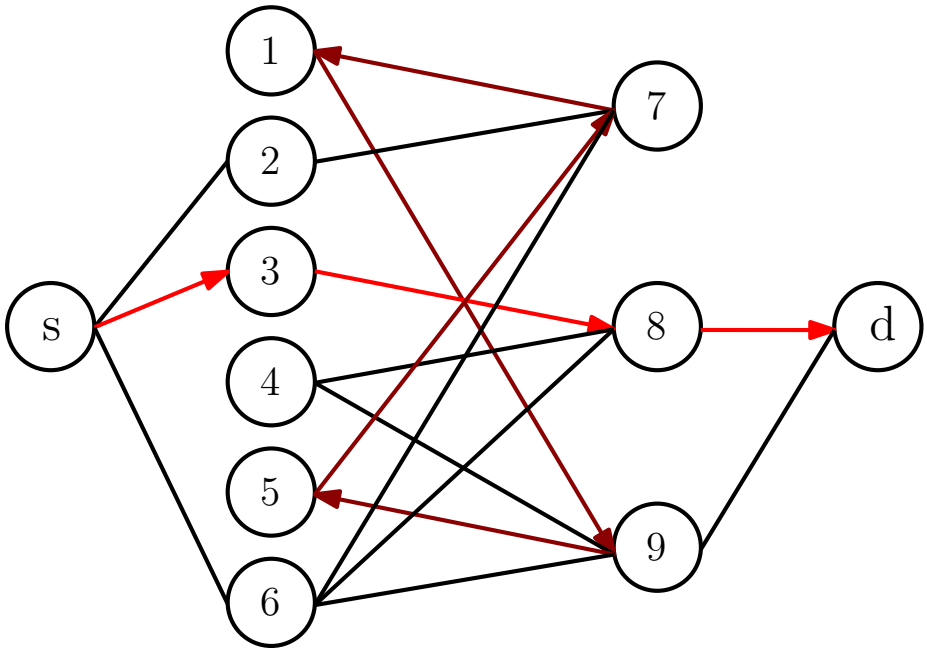


Figure 6.4: Construction of disjoint path-loop (mark with dark red) in a selected path.

where M is a large integer, and in our formulation we set it to the value $\lfloor \frac{|E|}{2} \rfloor$.

Now $\forall v_l \in L$ in the l^{th} selected path the sum of current value through all the incident edges at vertex v_l is equal to v_l. Similarly, $\forall v_r \in R$ in the l^{th} selected path the sum of current value through all the incident edges at vertex v_r is equal to the negative value of v_r. These constraints are formulated as

$$\begin{aligned} \sum_{\forall i \in L,\ \forall j \in R \cup \{s\}} f_{ij}^l &= v_i^l; \\ \sum_{\forall i \in R,\ \forall j \in L \cup \{d\}} f_{ij}^l &= -v_i^l; \end{aligned} \tag{6.5}$$

Figure 6.5, illustrate the removal of disjoint loop with the help of constraints 6.4 and 6.5. Here, for the l^{th} selected path, flow variable corresponding to the edges are assign integer values such that $f_{17}^l = 5$, $f_{19}^l = -4$, $f_{57}^l = -2$, $f_{59}^l = 3$. These four combination of flow variables satisfies three constraints where $v_1^l = v - 5^l = 1$ and $v_9^l = -1$. But, did not satisfy the constraints for v_7^l. Similarly, it can be shown that no combination of flow variables are able to satisfy all the four constraints for disjoint loop. Therefore, constraints

6.4 to 6.5 effectively prevent the disjoint path-loop to appear the the solution path.

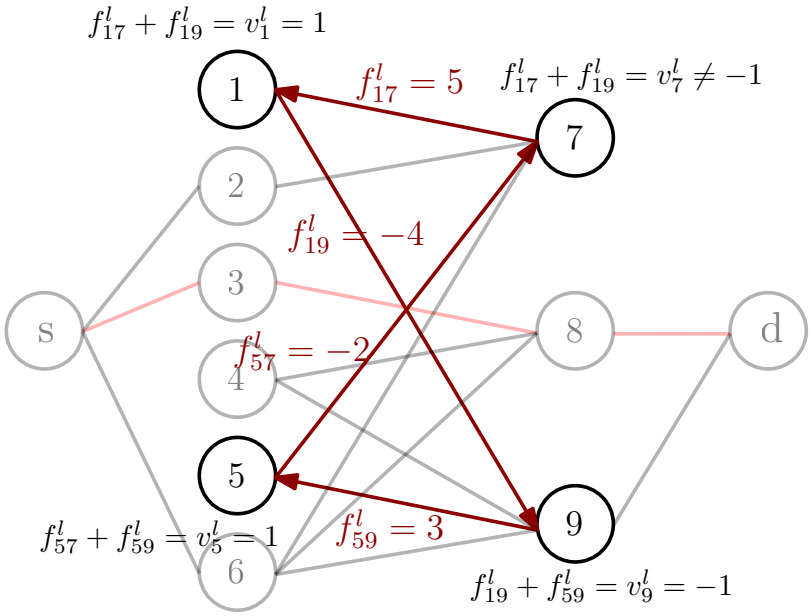


Figure 6.5: Removal of disjoint path-loop (mark with dark red) in a selected path.

6.3.3 Path minimization and formulation of the *ILP*

Our goal is to select a minimum number of paths from vertex s to d which satisfies all the constraints formulated above. In order to get the solution, we first consider a constant value of n_p (preferably starting from $n_p = 1$) and try to satisfy all the constraints. Since we assign a fixed value to n_p, it is possible that the formulated *ILP* has no solution implying that all the edges are not covered. Then we repeatedly increment n_p by one and try to solve the *ILP* until a valid solution is obtained. The minimum value of n_p which solves the *ILP* is our required solution. Therefore, our mixed ILP formulation effectively becomes a constraint satisfaction problem, rather than that of an optimization one.

For a fixed n_p, the complete *ILP* is given below.

$$n_p \quad a\ positive\ integer\ constant\ (objective\ function)$$

$$\sum_{\forall i \in L} e_{si}^{l} = 1;$$

$$\sum_{\forall i \in R} e_{id}^{l} = 1;$$

$$\sum_{\forall i \in L,\ \forall j \in R \cup \{s\}} e_{ij}^{l} = 2 * v_{i}^{l};$$

$$\sum_{\forall i \in R,\ \forall j \in L \cup \{d\}} e_{ij}^{l} = 2 * v_{i}^{l};$$

subject to

$$\sum_{l=1}^{n_p} e_{ij}^{l} \geq 1, \quad \forall\ e_{ij} \in E;$$

$$f_{ij}^{l} \leq M * e_{ij}^{l};$$

$$f_{ij}^{l} \geq -M * e_{ij}^{l};$$

$$\sum_{\forall i \in L,\ \forall j \in R \cup \{s\}} f_{ij}^{l} = v_{i}^{l};$$

$$\sum_{\forall i \in R,\ \forall j \in L \cup \{d\}} f_{ij}^{l} = -v_{i}^{l};$$

$$\forall l \in \{1, 2, \ldots, n_p\}.$$

Example 6.3.1. *Here, an complete ILP formulation for crossbar of size* 4×4 *(as shown in Figure) is presented. The corresponding solution with IBM ILOG CPLEX Optimisation Studio is also presented below.*

```
Maximize
obj: e12 + e13 + e14 + e25 + e26 + e27 + e35 + e36 + e37 + e45
+ e46 + e47 + e85 + e86 + e87

Subject To
e12 + e13 + e14 = 1
e85 + e86 + e87 = 1
e12 + e25 + e26 + e27 - 2 v2 = 0
e13 + e35 + e36 + e37 - 2 v3 = 0
e14 + e45 + e46 + e47 - 2 v4 = 0
e85 + e25 + e26 + e45 - 2 v5 = 0
e86 + e35 + e36 + e46 - 2 v6 = 0
e87 + e45 + e46 + e47 - 2 v7 = 0
```

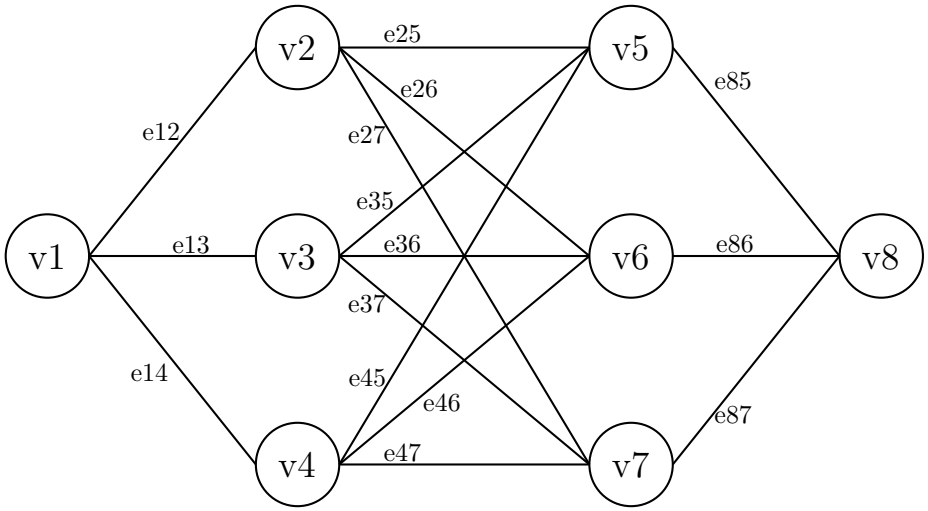


Figure 6.6: Graph G_{spn} for ILP formulation

```
f12 - 7 e12 <= 0;   - f12 - 7 e12 <= 0
f25 - 7 e25 <= 0;   - f25 - 7 e25 <= 0
f26 - 7 e26 <= 0;   - f26 - 7 e26 <= 0
f27 - 7 e27 <= 0;   - f27 - 7 e27 <= 0
f27 - 7 e85 <= 0;   - f27 - 7 e85 <= 0
f13 - 7 e13 <= 0;   - f13 - 7 e13 <= 0
f35 - 7 e35 <= 0;   - f35 - 7 e35 <= 0
f36 - 7 e36 <= 0;   - f35 - 7 e35 <= 0
f37 - 7 e37 <= 0;   - f36 - 7 e36 <= 0
f14 - 7 e14 <= 0;   - f14 - 7 e14 <= 0
f45 - 7 e45 <= 0;   - f45 - 7 e45 <= 0
f46 - 7 e46 <= 0;   - f46 - 7 e46 <= 0
f47 - 7 e47 <= 0;   - f47 - 7 e47 <= 0
f47 - 7 e87 <= 0;   - f47 - 7 e87 <= 0
f12 + f25 + f26 + f27 - v2 = 0
f13 + f35 + f36 + f37 - v3 = 0
f14 + f45 + f46 + f47 - v4 = 0
f85 + f25 + f26 + f45 + v5 = 0
f86 + f35 + f36 + f46 + v6 = 0
f87 + f45 + f46 + f47 + v7 = 0
```

```
Bounds
- 7 <= f12, f25, f26, f27, f85, f13, f35, f36, f37, f86, f14,
f45, f46, f47, f87 <= 7
Integers
f12, f25, f26, f27, f85, f13, f35, f36, f37, f86, f14, f45,
f46, f47, f87
Binaries
e12, e25, e26, e27, e85, e13, e35, e36, e37, e86, e14, e45,
e46, e47, e87, v2, v3, v4, v5, v6, v7
End

Solutions:
Total edge covered = 21
Extra edge covered = 6
The path list are following:
Path1: [‘e14’, ‘e86’, ‘e25’, ‘e27’, ‘e35’, ‘e36’, ‘e47’]
Path2: [‘e13’, ‘e85’, ‘e26’, ‘e27’, ‘e37’, ‘e45’, ‘e46’]
Path3: [‘e12’, ‘e87’, ‘e26’, ‘e35’, ‘e37’, ‘e45’, ‘e46’]
```

ILP formulation for G_{spn} not for the crossbar

In the proposed mixed *ILP*, each constraint is formulated for the graph G_{spn}, not for the memristive crossbar itself. For a full square or rectangular crossbar, the structure of the graph G_{spn} is invariant to the selection of memristors. But for an incomplete crossbar, the graph G_{spn} for the same particular crossbar is not fixed and varies with the choice of the selected WL and BL; hence the complexity of the *ILP* differs. In Figure 6.8 we have shown two different G_{spn} corresponding to the same incomplete crossbar based on two different selections of the WL and BL in the crossbar.

Very large number of constraints and variables

It is observed that for crossbar size greater than 16×16, solving *ILP* with either IBM ILOG CPLEX Optimization Studio or LINGO 16.0 Optimization Modeling Software takes a very long time, possibly due to a large number of constraints and variables in the *ILP*. Table 6.1

presents the number of constraints and variable of the ILP formulated for a square full crossbar of size $n \times n$.

Table 6.1: Number of constrainrs required for ILP formulation of a crossbar of size $n \times n$

Formulation of	**# Constraints**
Constraints 6.1	2
Constraints 6.2	$2(n-1)$
Constraints 6.3	n^2-1
Constraints 6.4	$2(n^2-1)$
Constraints 6.5	$2(n-1)$

Table 6.2: Number of variables required for ILP formulation of a crossbar of size $n \times n$

Variables	**Count**
Binary: Edges $e_{ij} \forall\ e_{ij} \in E$	n^2-1
Binary: vertices $v_i \forall i \in R \cup L$	$2(n-1)$
Integer: flow variavles f_{ij}	n^2-1

It is found from the theoretical proposition in Chapter 5 that $n_p = n-1$ is the minimum value that satisfies the ILP. Therefore, for an $n \times n$ square complete crossbar, this *ILP* has $(n-1)(2n-1)(n+3)$ constraints and $2(n-1)^2(n+2)$ variables in total. Plot 6.9 shows the number of constraints and variables required for different sizes of the crossbar.

6.4 Evaluation of the proposed methods

We have already mentioned in the previous chapter that our proposed heuristics provide the optimum (minimum) number of selected paths for full square and rectangular crossbars. In order to prove our proposition, we formulate *ILP* for full square crossbar of different sizes and in Table 6.3, we showed that the test time obtain from the proposed algorithm, and from the formulated *ILP* are the same. We run the *ILP* both in IBM ILOG CPLEX Optimization Studio as well as in LINGO 16.0 Optimization Modeling Software on a 4-core

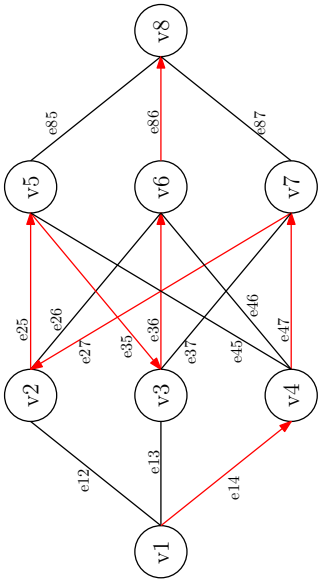
3GHz Intel Xeon processor with 32GB RAM and came up with the same result. It is clear from the Table 6.3 that the solution time for the *ILP* is increasing rapidly with crossbar size and both the solver is unable to solve the *ILP* formulated for the crossbar of size grater than 16×16 (may be due to a large number of constraints). Therefore, though both the proposed heuristics as well as the *ILP* is providing the same result in terms of solution time, our proposed heuristics are much more efficient.

Table 6.3: Comparison of the proposed method with ILP solution.

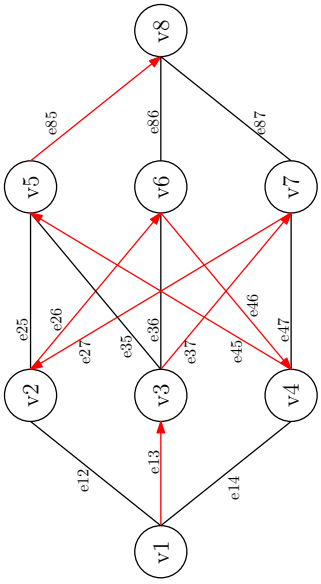
Crossbar size	Path count		Edge count		ILP solution time (s)
	TOSPS	ILP	TOSPS	ILP	
4×4	3	3	15	15	0.01
6×6	5	5	35	35	0.13
8×8	7	7	63	63	0.65
10×10	9	9	99	99	1.38
12×12	11	11	143	143	4.96
14×14	13	13	195	195	527.83
16×16	15	15	255	255	6521.79

6.5 Summary

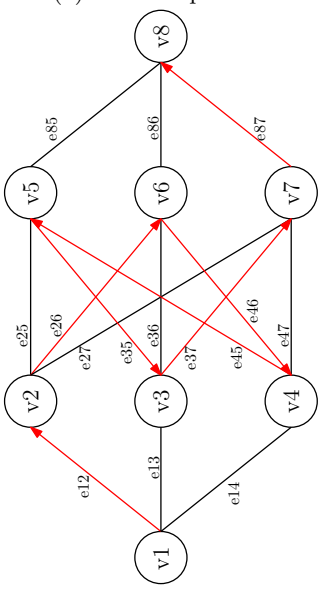
In this chapter, we present an Integer Linear Program (ILP) designed to tackle the path selection issue in an incomplete crossbar. While the current ILP solver works effectively for crossbars with sizes up to 16, its performance notably deteriorates when dealing with larger crossbars, resulting in longer computation times. To overcome this challenge, a promising direction could involve developing an ILP tailored for memristive crossbars of arbitrary sizes and shapes, with an optimal number of variables and constraints. Such an approach has the potential to enhance the efficiency and scalability of path selection in diverse crossbar configurations.



(a) Solution path-1



(b) Solution path-2



(c) Solution path-3

Figure 6.7: Three selected paths for the ILP using IBM ILOG CPLEX Optimisation Studio.

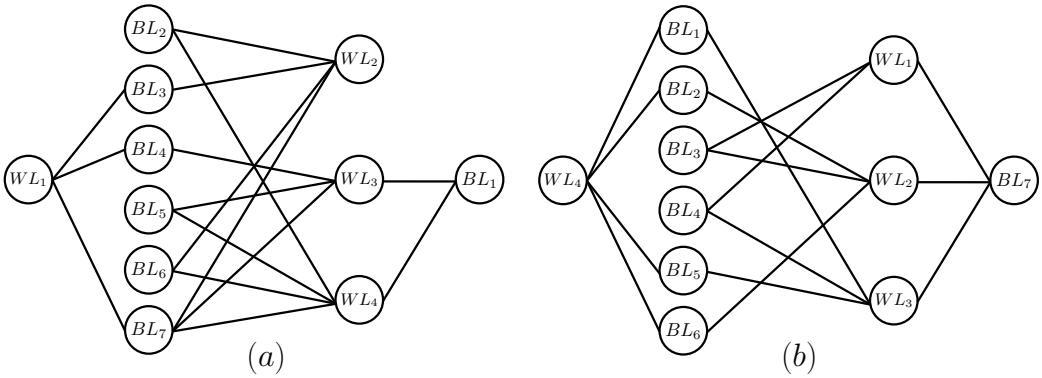


Figure 6.8: (a) The graph for *ILP* corresponding to the crossbar shown in 6.1(a) if input voltage is applied between WL_1 and BL_1, and (b) if input voltage is applied between WL_4 and BL_7.

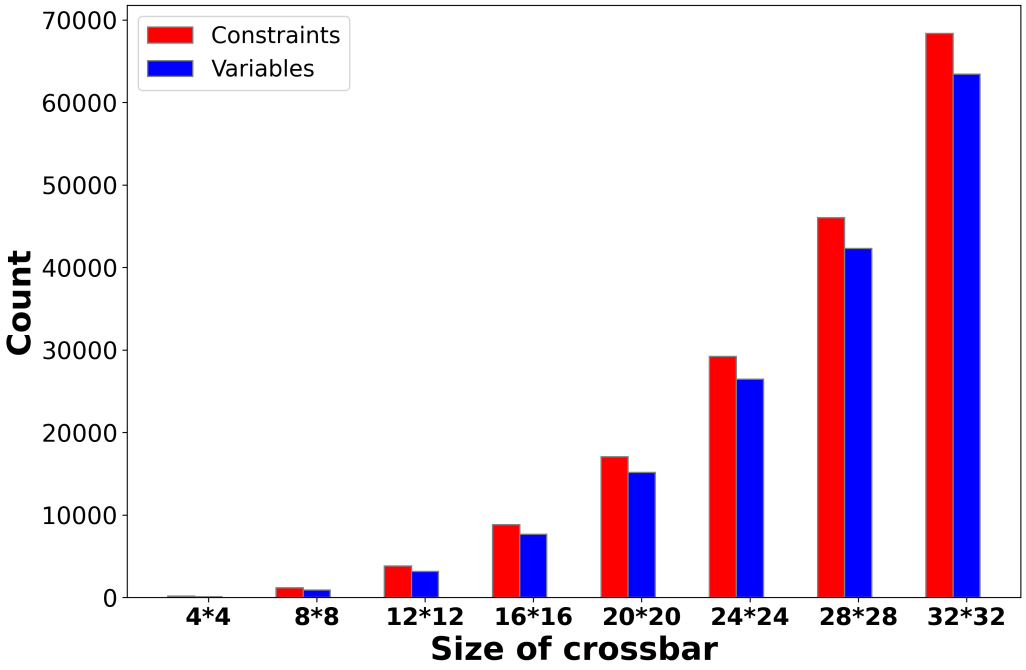


Figure 6.9: Number of constraints and variables required for different sizes of crossbar.

CHAPTER

SEVEN

ANALYSIS OF FAULT SENSITIVITY IN MEMRISTIVE NEURAL ARCHITECTURES

Contents

7.1 Introduction

Memristive crossba-based architectures have emerged as an emerging candidate for a new generation computing platform (HXZ+21; BGS+18; BSS+17). The storage along with the computing ability of the memristor element has the potential to overcome the memory-processor bottleneck of the Von Neumann architecture to a certain extent. The properties like non-volatility and continuous input/output characteristics of the memristor are ideal for analog computations. Memristor based crossbar (MBC) is very efficient in computing matrix-vector multiplication, vector outer product, etc. which are very costly operations in terms of space, time, and energy (PMBH+15). Memristive crossbar has the ability to perform massive parallel computations with minimum data movement. resulting less consumption of energy per computations than the traditional CMOS based accelerators (KJC20). Therefore, a lot of research work have been proposed to implement artificial neural network (ANN), recursive neural network (RNN), and spiking neural network (SNN) in memristive crossbar (SSS+20; ASKJ18; YAT17; WWZH18).

Due to the parallel and distributed structure, and the presence of additional neurons or processing elements than the necessary to solve a given problem, neural networks are assumed to be inherently fault tolerant (AJB00). However, it is very hard to generalize fault tolerance assessment across different neural architectures as well as the architectures made of different computing elements as there is no systematic methods and tools for evaluation across neural models. Due to the CMOS-memristor heterojunction fabrication, MBCs suffers from various defects and faults, leading to a significant accuracy loss and errors in neural networks (ZUFE20; XCG+20; CC22; SSB+22). Manufacturing faults are permanent and make the memristor get stuck at high or low resistance state. There are other types of faults which occur due to inaccurate writing of memristor cells while data processing. This are called soft faults (DFR+15). The occurrence of the hard and soft faults in the memristor cell of MBC limits the accuracy of the MBC based computing system from reaching to its maximum achievable value (PMBH+15).

In this chapter, we conducted a comprehensive analysis of the effect of stuck-at-zero ($SA0$) and stuck-at-one ($SA1$) faults on the accuracy degradation of an MNN consisting of two convolutional layers, two fully connected layers, and a memristive LeNet network. In the course of our simulations, we opted for the utilization of an open-source framework

specifically designed for conducting large-scale memristive Deep Learning (DL) simulations, known as MemTorch (LXLBR22). Within the confines of this framework, we undertook the task of generating a fault model, and further tailored a fault injection technique to suit the specific requirements of our study. This area of research is particularly important because understanding the influence of such faults and variations can contribute to the development of fault-tolerant neuromorphic computing systems with limited resources. Additionally, by gaining insight into the impact of faults on individual layers of memristive crossbar-based neural architectures, we may discover optimal architecture designs for solving specific problems with predefined accuracy.

7.2 Preliminaries

7.2.1 Neural networks

Memristive neural models are artificial neural networks that utilize memristors as a computing as well as processing elements. Memristive neural models have been proposed as a potential alternative to traditional artificial neural networks, which are typically implemented using electronic circuits or software. There are several different approaches that have been proposed for implementing memristive neural models, including both hardware-based and software-based approaches. Hardware-based approaches typically involve designing circuits or systems that utilize memristors to implement the neural network, while software-based approaches typically involve developing algorithms that can be implemented on traditional computing platforms and simulate the behavior of a memristive neural network.

7.2.2 Faults in neural models

One major challenge with MNNs is the issue of faults, which can occur due to various factors such as manufacturing defects, wear and tear, or external factors such as temperature. Faults in MNNs can lead to incorrect or unreliable operation, which can significantly degrade the performance of the network.

Stuck-at-one (SA1) faults and stuck-at-zero (SA0) faults are common types of hard faults that can occur in memristors. SA1 faults are caused by permanent open switch defects and broken word-lines, while SA0 faults are caused by permanent short circuit defects and broken bit-lines. Both SA1 and SA0 faults can significantly impact the performance of a memristive neural network (MNN) and can lead to incorrect or unreliable operation. A

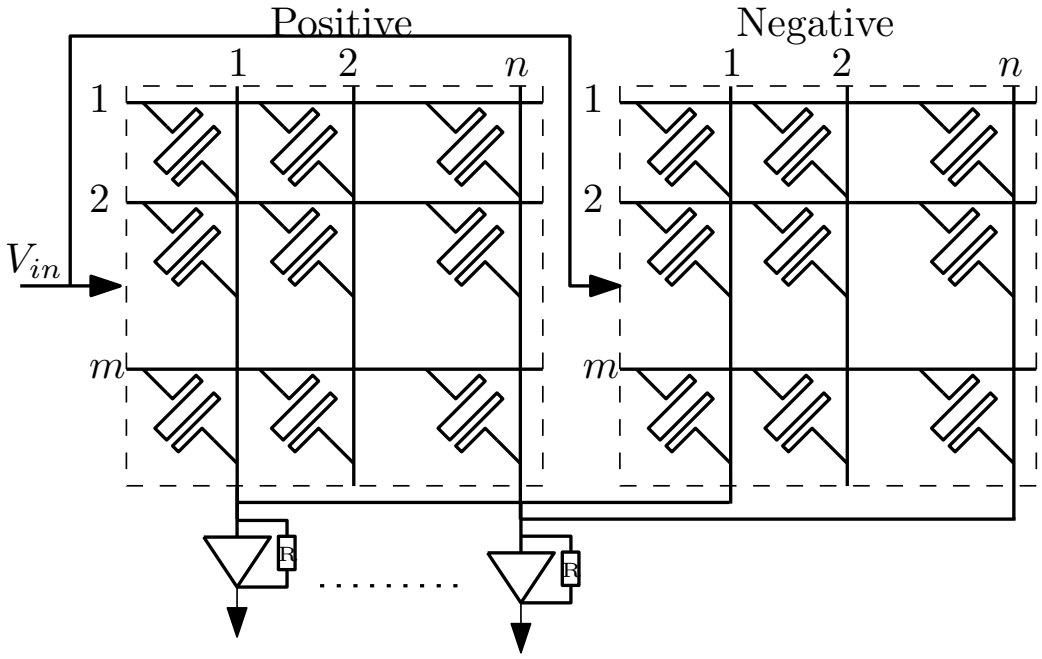


Figure 7.1: Memristor based crossbar using a differential pair of memristor for representing a weight value.

memristor device with SA1 fault is always in the high resistance state (HRS), whereas SA0 faults force the on-chip memristors in a low resistance state (LRS).

Stochastic variation in memristors refers to the inherent uncertainty or randomness in the behavior of these devices. Memristors are passive two-terminal devices that can change their resistance based on the amount of current that has passed through them. However, the resistance of a memristor can also vary due to random fluctuations in the device's physical properties, such as the mobility of dopants or impurities within the device. Stochastic variation in memristors can impact the performance of a memristive neural network (MNN) by introducing uncertainty into the operation of the network. This can lead to variations in the output of the network, which can affect its accuracy and reliability.

7.2.3 Basics of fault-tolerance

Here are some terms related to fault tolerance:

- **Reliability**: Reliability is an important characteristic of any system, as it determines

the likelihood that the system will perform correctly and meet the required specifications over a specified period of time. Reliability can be affected by various factors, including the quality of the components used in the system, the design of the system, and the operating conditions. Systems with high reliability are often designed with built-in redundancies and failover mechanisms to ensure that they can continue to operate correctly even in the event of failures or faults.

- **Fault tolerance**: Fault tolerance is the ability of a system to continue functioning correctly despite the occurrence of faults or failures within some of its components. This means that the system is able to withstand or recover from these faults without disrupting its normal operation. Fault tolerance can be achieved through various methods, such as using redundant components, implementing failover mechanisms, and designing the system to be resilient to failures.
- **Robustness**: Robustness refers to the ability of a system to maintain its performance and functionality despite variations in its inputs or parameters. A system that is robust is able to tolerate noise or uncertainties in its inputs and still produce the desired output. This is important in situations where the system may be subjected to variations in its operating conditions, such as changes in temperature, humidity, or other environmental factors.
- **Error resilience**: Error resilience is the ability of a system to tolerate inexact or approximate computations without affecting its overall performance or functionality. This means that the system is able to continue operating correctly even when some of the computations or calculations being performed are not exactly accurate. Error resilience is often important in situations where the system may be subject to noise or other sources of error, such as when working with data that has been transmitted over a noisy channel or when performing calculations with limited precision.

7.2.4 Active and passive fault-tolerance

In the context of fault tolerance, fault tolerance can be classified into passive and active, depending the mechanisms by which it is achieved in a system.

Active fault tolerance refers to approaches that actively detect and mitigate the effects of faults or failures in a system. This can involve using redundant components or systems that can take over the function of a failed component, implementing failover mechanisms to switch to a backup system when a primary system fails, or using other proactive measures to prevent or mitigate the impact of failures.

On the other hand, passive fault tolerance refers to approaches that do not actively detect or mitigate the effects of faults, but rather rely on the system's inherent resilience to continue functioning correctly despite the occurrence of faults. Passive fault tolerance approaches may involve designing the system to be resilient to failures, using redundant components or systems to increase reliability, or other methods that allow the system to continue operating correctly in the event of a failure. In such passive fault tolerant system, no diagnostics, relearning, or reconfiguration is required. Thus, fault detection and location can be totally avoided under this approach.

Both active and passive fault tolerance approaches can be effective in increasing the reliability and fault tolerance of a system, and the appropriate approach will depend on the specific requirements and constraints of the system.

7.3 Problem Formulation

Memristive devices can fail or experience errors due to various factors such as manufacturing defects, environmental conditions, or wear and tear. This can have a significant impact on the performance and reliability of the neuromorphic computing system. Therefore, it is important to conduct fault analysis to identify and diagnose any potential issues with the memristive devices and take corrective action to ensure the system is operating correctly.

Tile architecture is a type of design used in memristive crossbar arrays, which are a common building block in neuromorphic computing systems. In a tile architecture, the crossbar array is divided into smaller units, called tiles, which can be individually addressed and controlled. Each tile consists of a group of memristive devices arranged in a row and column configuration, with one row and one column of devices per tile. The tiles are connected together to form the larger crossbar array. The advantage of using a tile architecture in a memristive crossbar is that it allows for more flexible and scalable designs. It allows for the independent control of each tile, enabling the system to perform more complex computations and tasks. It also allows for easier maintenance and repair, as faulty or damaged tiles can be replaced without affecting the rest of the system. Overall, the tile architecture is an important design feature in memristive crossbar arrays, providing flexibility, scalability, and maintenance benefits for neuromorphic computing systems. In order to design a neuromorphic computing system, there is no systemetic approach for selecting the tile size. Our aim is to select tile size depending on the stochastic variation of the weight values in the memristive crossbar.

In this chapter, we explore fault simulation, which involves using computer simulations to determine the impacts on system performance, including:

1. Evaluating the natural fault tolerance of an individual layer to injected SAFs.
2. Identifying which layer of a memristive neural network has the greatest impact on accuracy in the presence of SAFs.
3. Determining which layer is most sensitive to the location of injected faults.
4. Examining whether $SA0$ and $SA1$ doses have the same effect on the network.

Answers to these questions are significant as understanding the influence of faults and variations can contribute to research areas such as determining the tile size for each layer, designing fault-tolerant MNN with limited resources and searching for optimal architecture designs to solve specific problems with predefined accuracy.

7.4 Overview of the proposed method

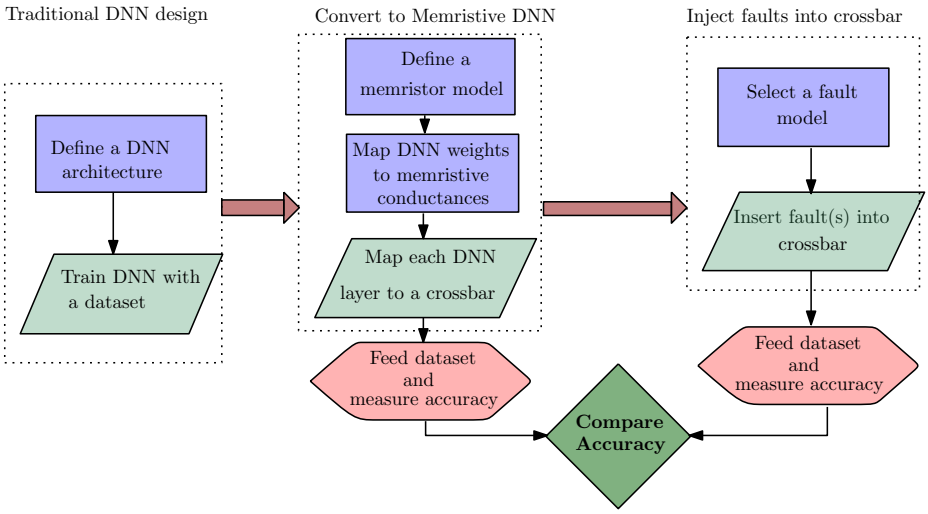


Figure 7.2: Flowchart for our method for Analysis of Fault Sensitivity

Figure 7.2 illustrates the design flow for implementing a MNN using the PyTorch framework and MNIST dataset. Initially, a DNN architecture is designed and trained using PyTorch and the MNIST dataset in the first block. Then, the trained ANN is converted into a MNN in the second block. This conversion involves replacing each weight value with the conductance difference of two memristor devices, as shown in Figure 7.1. The weight-

to-conductance mapping rule is used to map each trained layer to a memristive crossbar, and the accuracy of the MNN framework is evaluated.

In the third block, SAFs are intentionally inserted into the memristive crossbar at different locations, and the accuracy of the faulty MNN is measured. Finally, the accuracy of the second and third blocks is compared to determine the effect of the inserted faults. The impact of the fault insertion is discussed further below.

To account for the random nature of fault insertion in crossbars, it is essential to conduct multiple runs when validating a defective MNN. Even with a fixed fault percentage, the locations of faults in the crossbars can vary in each run. Running the MNN multiple times helps to verify its performance, ensuring robustness, generalizability, and avoiding overfitting. To validate our experiments, we conducted ten runs for each fault percentage, and the resulting plots are presented in Section 7.4.

7.5 Experimental Evaluation

Our experiments were conducted on a 4-core 3GHz Intel(R) Xeon(R) Platinum 8164 CPU @ 2.00GHz processor with 32GB RAM, running Ubuntu 22.04 LTS. All simulations were performed using the MemTorch platform, and the VTEAM memristor model was used to convert trained architectures into MNNs. For simulation, we have taken two MNNs as follows:

1 . A memristive deep neural network (MDNN) containing two convolutional and two fully connected layers.

2. LeNet 5 architecture containing three convolutional and two fully connected layers.

Table 7.1 presents the datasets used for training two MNNs, along with the number of trainable parameters at each layer used for experimental evaluation. To observe the maximum impact of faults, we plotted the accuracy drop with an increasing fault percentage, taking the average accuracy value among all ten simulations. We varied the fault percentage from zero to ninety percent, although practical circuit faults typically remain below 25%. We included the full range of fault percentages for analysis purposes to obtain a comprehensive spectrum of accuracy drops.

Table 7.1: MDNN architectures with trainable parameters of all the layers and the datasets used

Neural Architecture	Trainable parameters					Datasets
	First Convolution layer (conv1)	Second Convolution layer (conv2)	First fully connected layer (fc1)	Second fully connected layer (fc2)	Third fully connected layer (fc1)	
MDNN	520	25050	400000	5000	NA	MNIST & FashionMNIST
LeNet 5	156	24164	48120	10164	850	MNIST & FashionMNIST

7.5.1 Effect of SA0 faults on accuracy

In order to demonstrate the influence of randomly inserting $SA0$ faults in each layer on accuracy, the subsequent section showcases the results. The accuracy is plotted against varying percentages of fault injection, focusing on one layer at a time. To assess the true impact, ten simulations were conducted for each layer at every fault percentage. Subsequently, the mean of the ten accuracy values was plotted to provide a comprehensive overview.

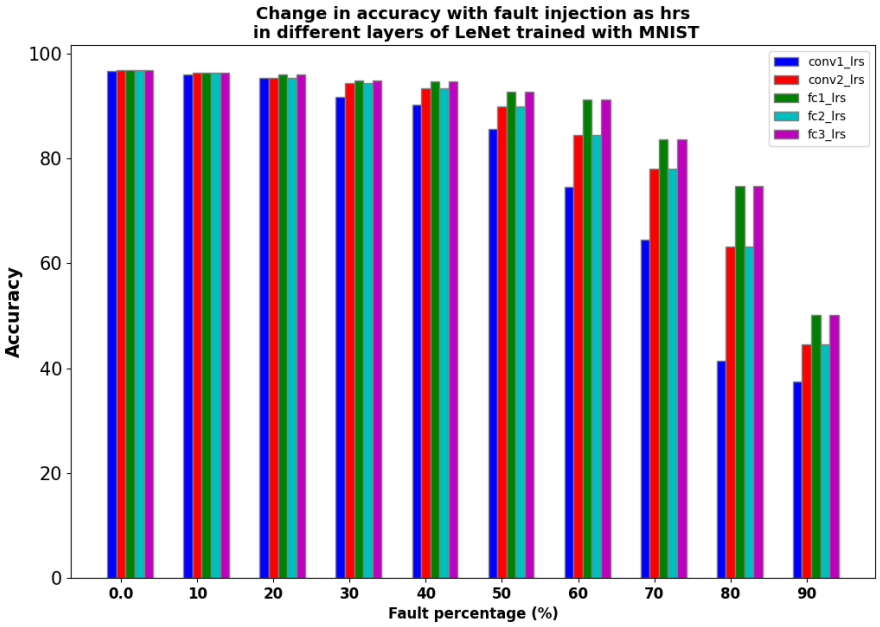
LeNet architecture trained with MNIST, and Fashion MNIST datasets

The accuracy plots in Figure 7.3a for each layer in the LeNet architecture trained with MNIST data reveal that the $conv1$ layer exhibits the most significant decrease in accuracy. The decline in accuracy becomes notable when the percentage of injected faults exceeds twenty. Both $conv2$ and $fc2$ layers have a similar impact on accuracy, slightly lower than that of $conv1$, with the accuracy drop occurring after injecting 40% of faults. The two linear layers, $fc1$ and $fc3$, have minimal effects on accuracy, showing a negligible impact until 60% of faults are injected.

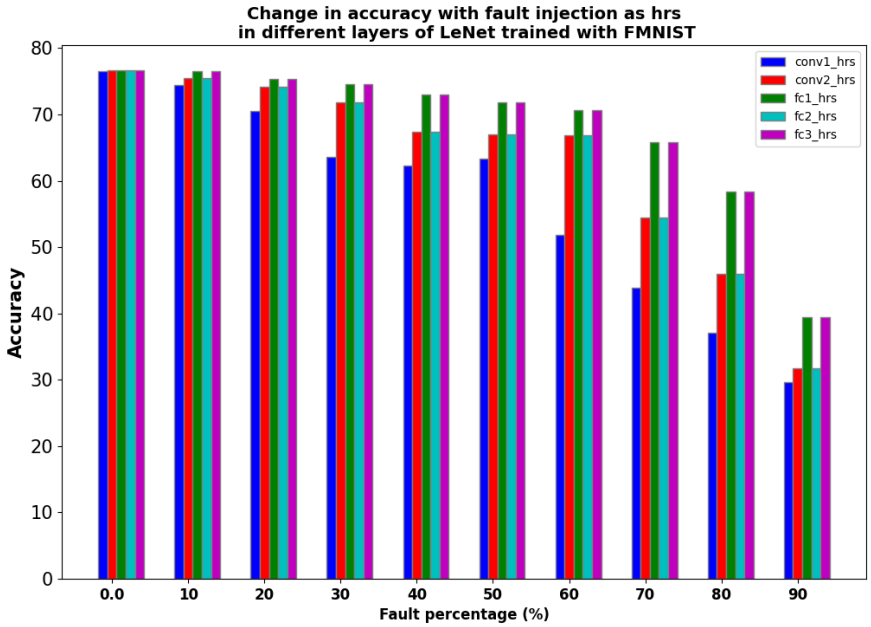
The plots in Figure 7.3b for each layer in the LeNet architecture trained with Fashion MNIST data exhibit a similar pattern, with slightly lower values of accuracy for all fault percentages compared to MNIST dataset. However, upto 40% fault injection mark, the impact on accuracy for all the layers is insignificant.

MDNN architecture trained with MNIST and Fashion MNIST datasets

The accuracy plots depicted in Figure 7.4ashowcase the impact on accuracy for each layer in the MDNN architecture trained with MNIST data. The results reveal that the maximum



(a) LeNet trained with MNIST



(b) LeNet trained with Fashion MNIST

Figure 7.3: Effect on Accuracy of randomly located varying number *hrs* ($SA0$) hardware faults on each layer of Memristive LeNet architecture trained on (a) MNIST and (b) Fashion MNIST.

drop in accuracy occurs in the $conv1$ layer, followed by the $fc1$ layer. On the other hand, the accuracy of the $conv2$ and $fc2$ layers is minimally affected, with nearly equal accuracy values observed for each fault percentage.

However, Figure 7.4b displays a different behavior for each layer in the MDNN architecture trained with Fashion MNIST data. Upto 40% of faulty memristors, the accuracy of the $conv1$ layer is the lowest. Beyond that point, all the three remaining layers experience lower accuracy, with the $fc1$ layer displaying the minimum value.

7.5.2 Effect of SA1 faults on accuracy

This section demonstrates the consequences of introducing $SA1$ faults randomly across all layers. Following the previous section, ten simulations were conducted for each layer at every fault percentage. Subsequently, the mean of the ten accuracy values was plotted to provide a comprehensive overview.

LeNet architecture trained with MNIST, and Fashion MNIST datasets

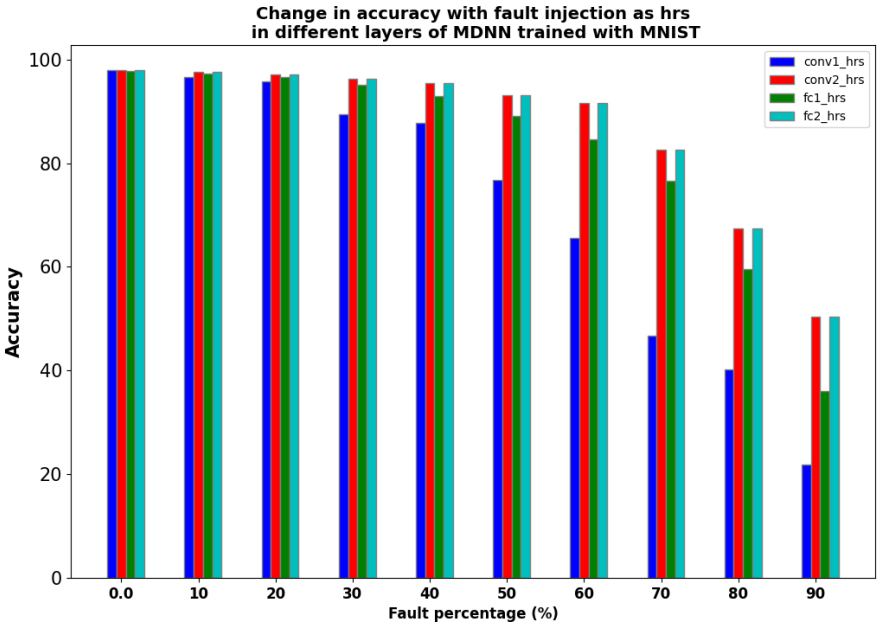
Figure 7.5a illustrates the accuracy plots for each layer in the LeNet architecture trained with MNIST data. The plots reveal that all layers exhibit a high sensitivity to injected faults, with a decrease in accuracy observed even at a minimal fault rate of 0.2%. Notably, layers $conv1$, $conv2$, and $fc2$ display the most significant impact on the introduced faults. Among them, the $conv1$ layer has the lowest accuracy for upto 40% faulty memristors, followed by $conv2$ and $fc2$ layers. Conversely, layers $fc1$ and $fc3$ exhibit minimal effects on the injected faults.

The accuracy plots for each layer in the LeNet architecture trained with MNIST data, depicted in Figure 7.5b, demonstrate similar trends as observed in case of MNIST dataset.

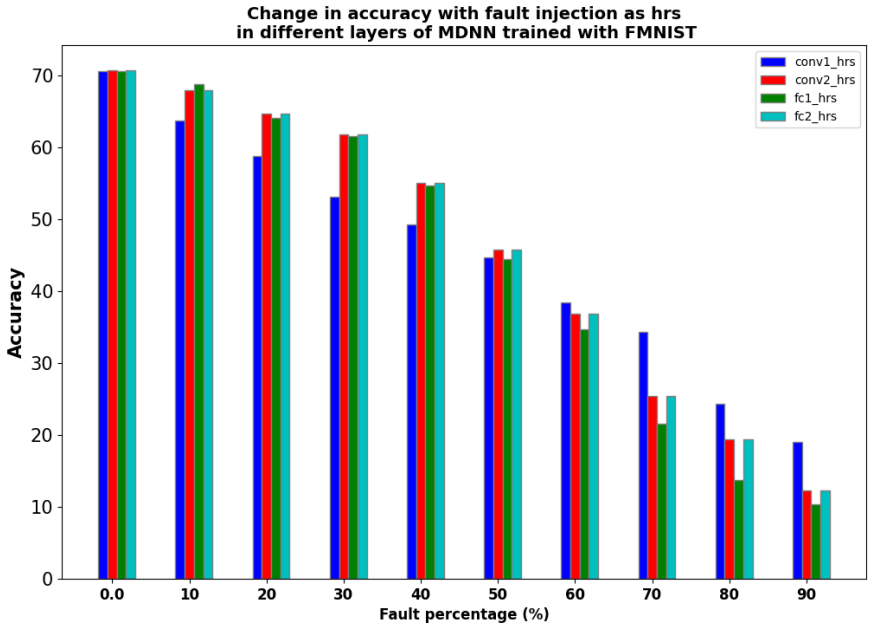
MDNN architecture trained with MNIST and Fashion MNIST datasets

Figures 7.6a and 7.6b illustrate the ranges of accuracy, which highlight the influence of each layer within the MDNN architecture trained on MNIST and Fashion MNIST datasets respectively. The findings demonstrate that the most significant decrease in accuracy is observed in the layer $conv1$, while the impact on accuracy in the $fc1$ layer is minimal for both the cases.

An interesting observation arises from comparing the impact of $SA1$ and $SA0$ faults across all four scenarios. Notably, the effect of $SA1$ faults differs considerably from that

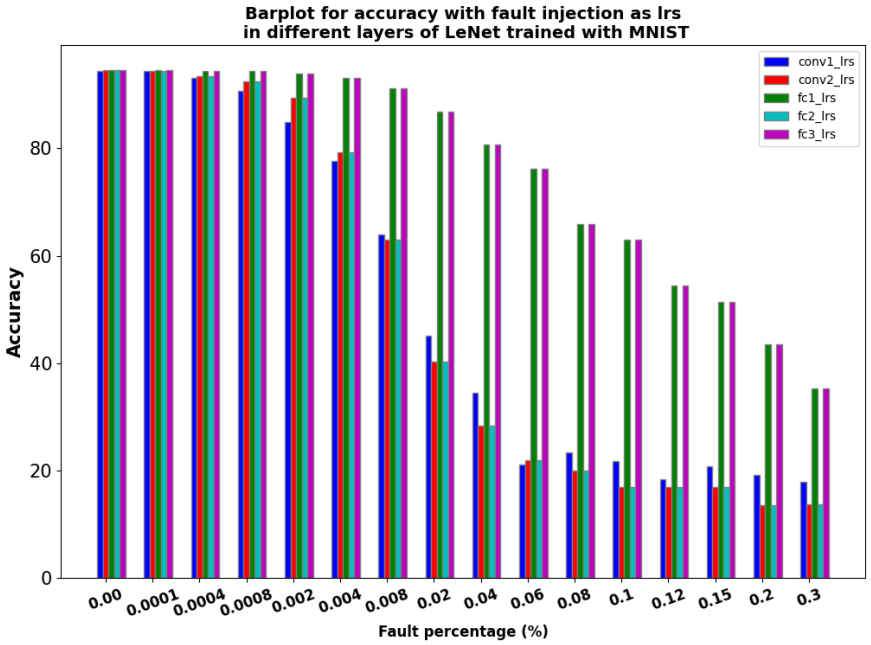


(a) MDNN trained with MNIST

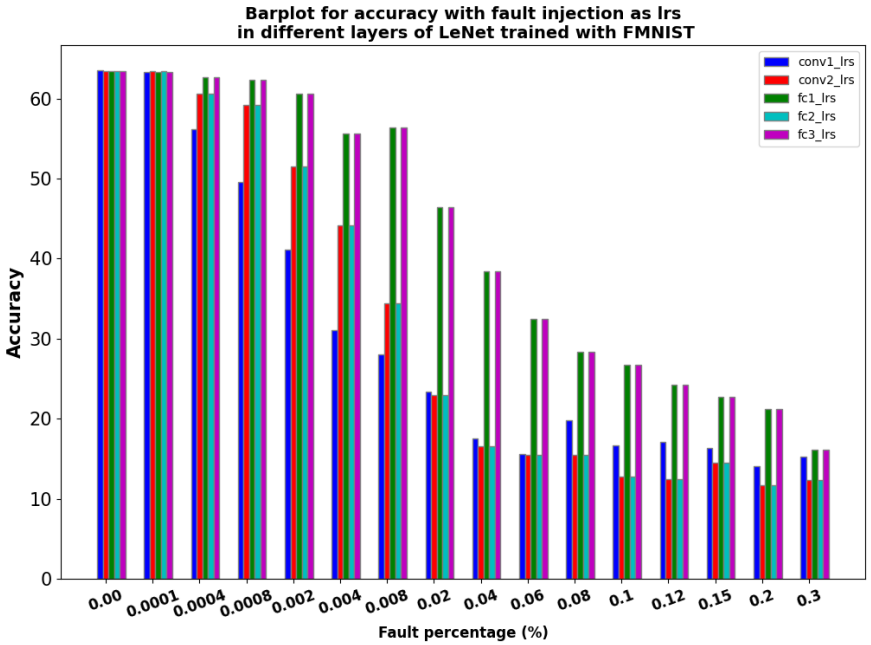


(b) MDNN trained with Fashion MNIST

Figure 7.4: Effect on Accuracy of randomly located varying number hrs ($SA0$) hardware faults on each layer of MDNN architecture trained on (a) MNIST and (b) Fashion MNIST.

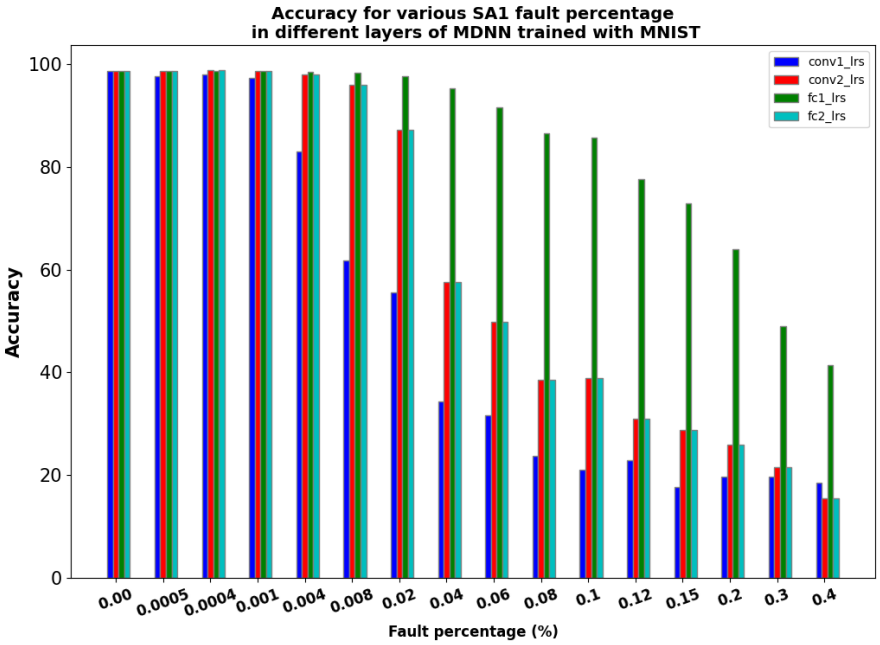


(a) LeNet trained with MNIST

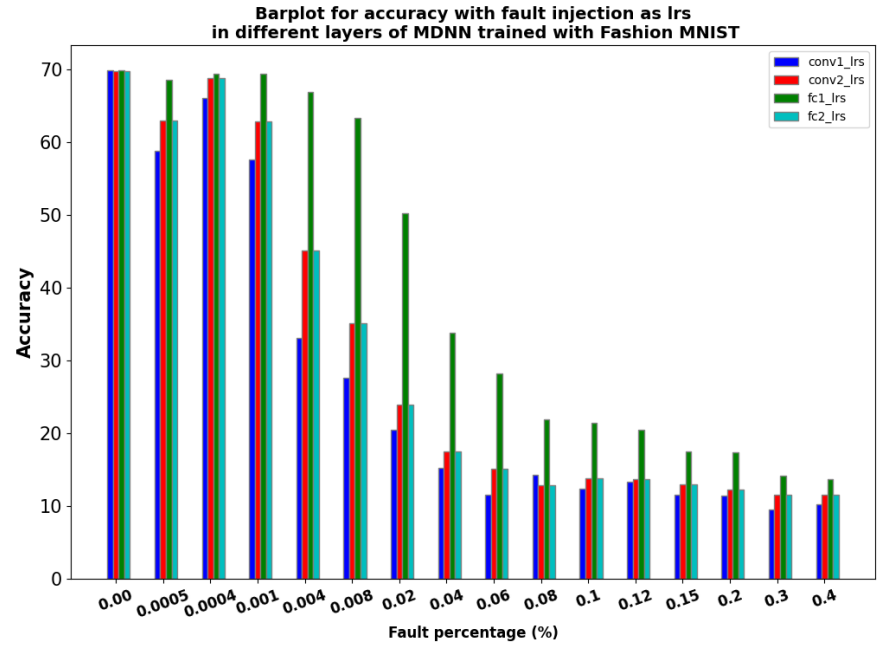


(b) LeNet trained with Fashion MNIST

Figure 7.5: Effect on Accuracy of randomly located varying number lrs ($SA1$) hardware faults on each layer of LeNet architecture trained on (a) MNIST and (b) Fashion MNIST.



(a) MDNN trained with MNIST



(b) MDNN trained with Fashion MNIST

Figure 7.6: Effect on Accuracy of randomly located varying number lrs ($SA1$) hardware faults on each layer of MDNN architecture trained on (a) MNIST and (b) Fashion MNIST.

of $SA0$ faults. In the case of high resistance states (hrs) injection, a significant drop in accuracy becomes apparent starting from a fault rate of 20%. Conversely, in the case of low resistance states (lrs), the accuracy experiences a noticeable decline when the fault rate exceeds approximately one percentage. This discrepancy presents an intriguing avenue for further investigation and exploration.

7.5.3 Layer-wise sensitivity analysis of LeNet architecture to fault locations

In order to observe the sensitivity to the fault location of SAFs for a particular layer, we have presented the box plots of fault simulation for each layer in this section. Figure 7.7 showcases the layer-wise sensitivity of $SA0$ faults at various fault percentages for all the layers. It becomes evident that the $conv1$ layer (Figure 7.7(a)) exhibits the highest sensitivity to fault location, as indicated by its maximum accuracy distribution compared to other layers. As the fault value surpasses the 40% threshold, the accuracy distribution demonstrates a noticeable increase.

It is noteworthy that layer $conv1$ consistently exhibits the highest sensitivity across all cases. Consequently, Figure 7.8 showcases the accuracy distribution of the $conv1$ layer in the LeNet architecture trained on both the MNIST and Fashion MNIST datasets. Figures 7.8(a) and (b) illustrate that the range of the distribution expands as the fault percentages increase. In contrast, Figure 7.8(c) highlights that the sensitivity is greater at lower fault percentages.

7.5.4 Layer-wise sensitivity analysis of MDNN architecture to fault locations

Figure 7.9 depicts the accuracy distribution of the MDNN architecture trained on the MNIST and Fashion MNIST datasets, considering the injection of both $SA0$ and $SA1$ faults. In Figure 7.9(a), the range of the distribution expands as the fault percentage increases. Conversely, in Figure 7.9(c), the distribution remains relatively uniform throughout the range of faults. Notably, Figures 7.9(b) and (d) indicate that the range of the distribution is highest at lower fault percentages and decreases as the fault percentages increase.

7.5.5 Distribution of weights in trained LeNet architecture

To gain insights into the behavior of the trained LeNet network when confronted with $SA0$ faults, we examined the distribution of weight values in the LeNet network trained on the MNIST and Fashion MNIST datasets. Figure 7.10 showcases these weight distributions.

Surprisingly, despite the weight distribution being mostly centered around a middle value, there is no apparent indication of the significant loss of accuracy observed in the network.

7.5.6 Discussion on Experimental Results

In this study, we conducted experiments to investigate the impact of fault injection on accuracy of two memristive CNN architectures trained using the MNIST and Fashion MNIST datasets. The results revealed that the $conv1$ layer is most susceptible to faults, indicating its vulnerability in the presence of errors. Additionally, we observed that among the linear layers, the one positioned right after the output layer displays the highest sensitivity in terms of accuracy. Moreover, a significant observation emerged, highlighting the differing impact of $SA0$ and $SA1$ faults, with $SA1$ faults exhibiting much greater sensitivity across all cases. An efficient misclassification-driven training (MDT) algorithm for detecting critical faults (FCFs) in the crossbar is proposed in (CC22). The algorithm successfully identifies the number of CFCs in each deep neural network (DNN) layer. However, the specific role of fault type as a location for being a CFC is not explored in the study. The researchers in (SFP+23) have introduced a fault injection platform that allows BNNs execution on logic-in memory while injecting in-field faults. While they extensively studied the impact of these faults on individual layers and different BNN models, the study does not explore the effects on general DNNs. Furthermore, we explored the sensitivity of fault location and found that accuracy variations were widespread. At times, the fluctuations exceeded 1.5 times the median value in ten simulations. Remarkably, this variation occurred randomly across architectures, datasets, and fault percentages, indicating the complex and unpredictable nature of the fault effects. It is important to note that all the observed effects were solely attributed to fault injection, as we did not find any discrepancies in the weight distribution of the trained architecture. This further confirms that the faults themselves were responsible for the accuracy loss in the memristive neural architectures under investigation.

7.6 Concluding Remarks

Examining faults in the crossbar of memristive neural architectures is a crucial undertaking in ensuring the reliability and robustness of these systems. To tackle this challenge, a fault analysis technique has been devised to determine the impact of stuck-at faults in different layers of LeNet neural architectures. This approach can evaluate the effect of injected faults on both the entire system and individual layers. The experimental results reveal that

the $conv1$ layer is most affected by fault injection, while among the linear layers, the one immediately behind the output layer shows the highest sensitivity in terms of accuracy; moreover, the impact of $SA1$ faults is considerably greater than $SA0$ faults. Overall, fault analysis in the crossbar of memristive neural architectures has the potential to contribute to research areas such as creating fault-tolerant MDNNs with limited resources and searching for optimal architecture designs to solve specific problems with predetermined accuracy.

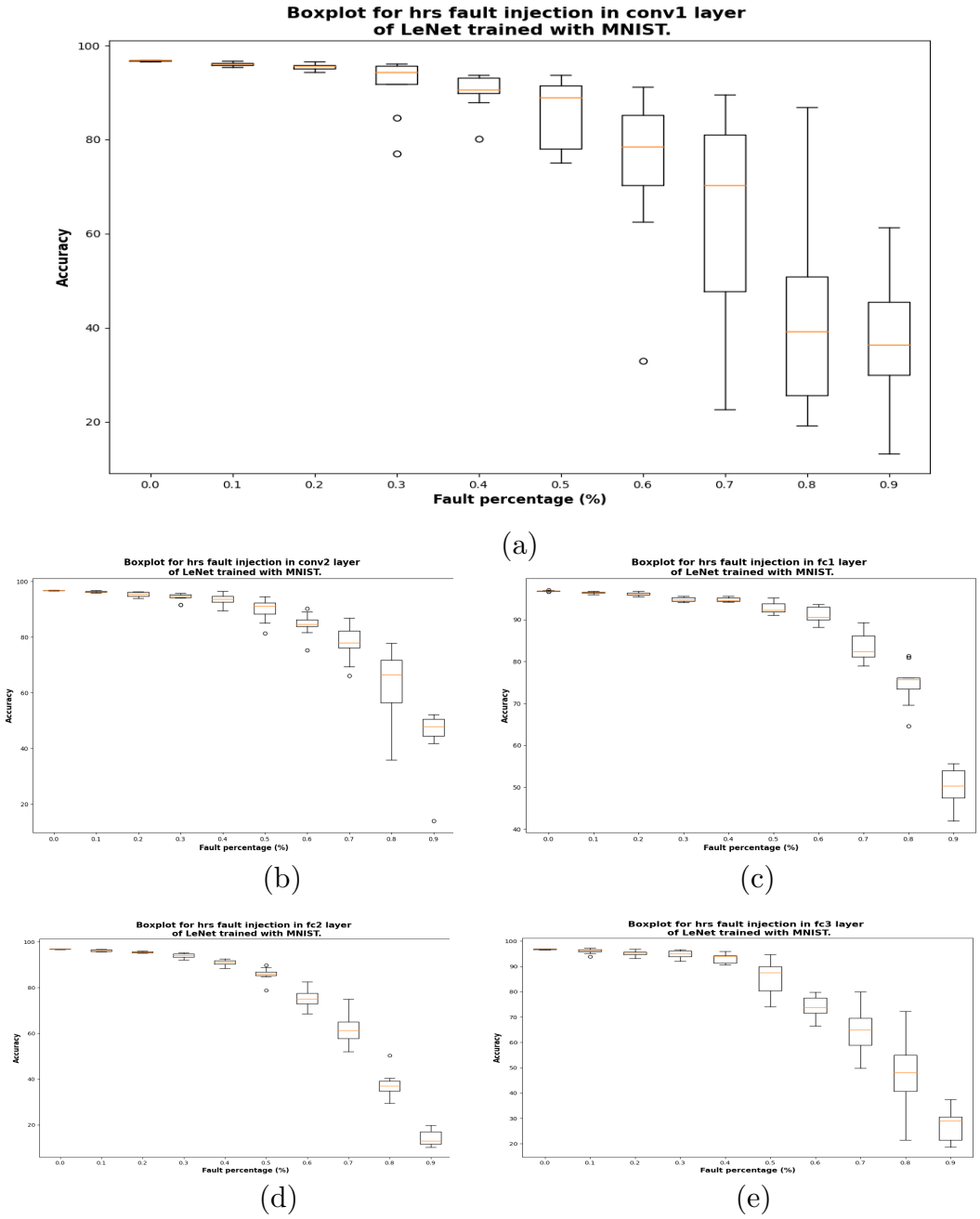
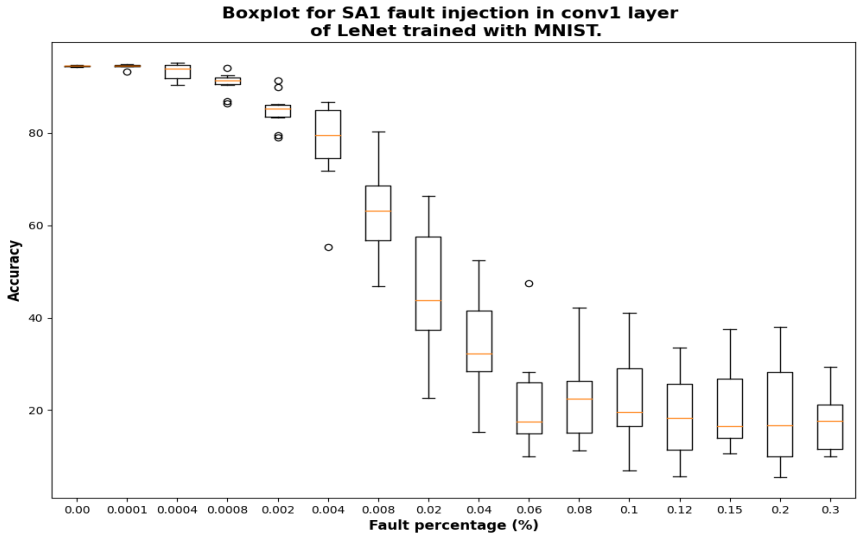
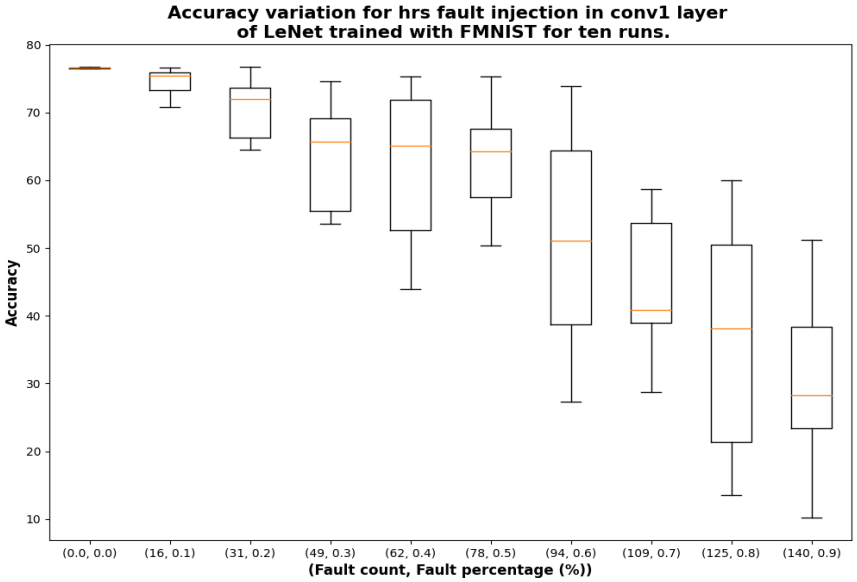


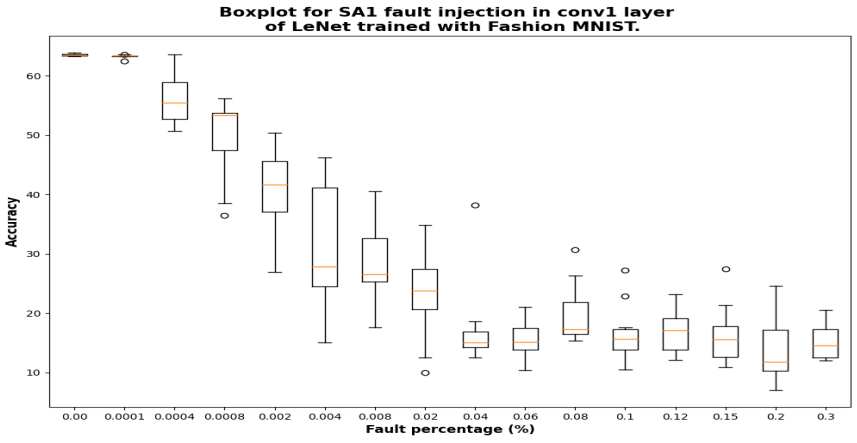
Figure 7.7: Accuracy range for ten simulations with *hrs* ($SA0$) fault injection in (a) $conv1$, (b) $conv2$, (c) $fc1$, (d) $fc2$, and (e) $fc3$ layer of the LeNet architecture trained with MNIST dataset.



(a)



(b)



(c)

Figure 7.8: Accuracy range for ten simulations of LeNet architecture trained for (a) MNIST with lrs ($SA1$) fault injected in $conv1$, (b) Fashion MNIST with hrs ($SA0$) fault injected in $conv1$, and (c) Fashion MNIST with lrs ($SA1$) fault injected in $conv1$.

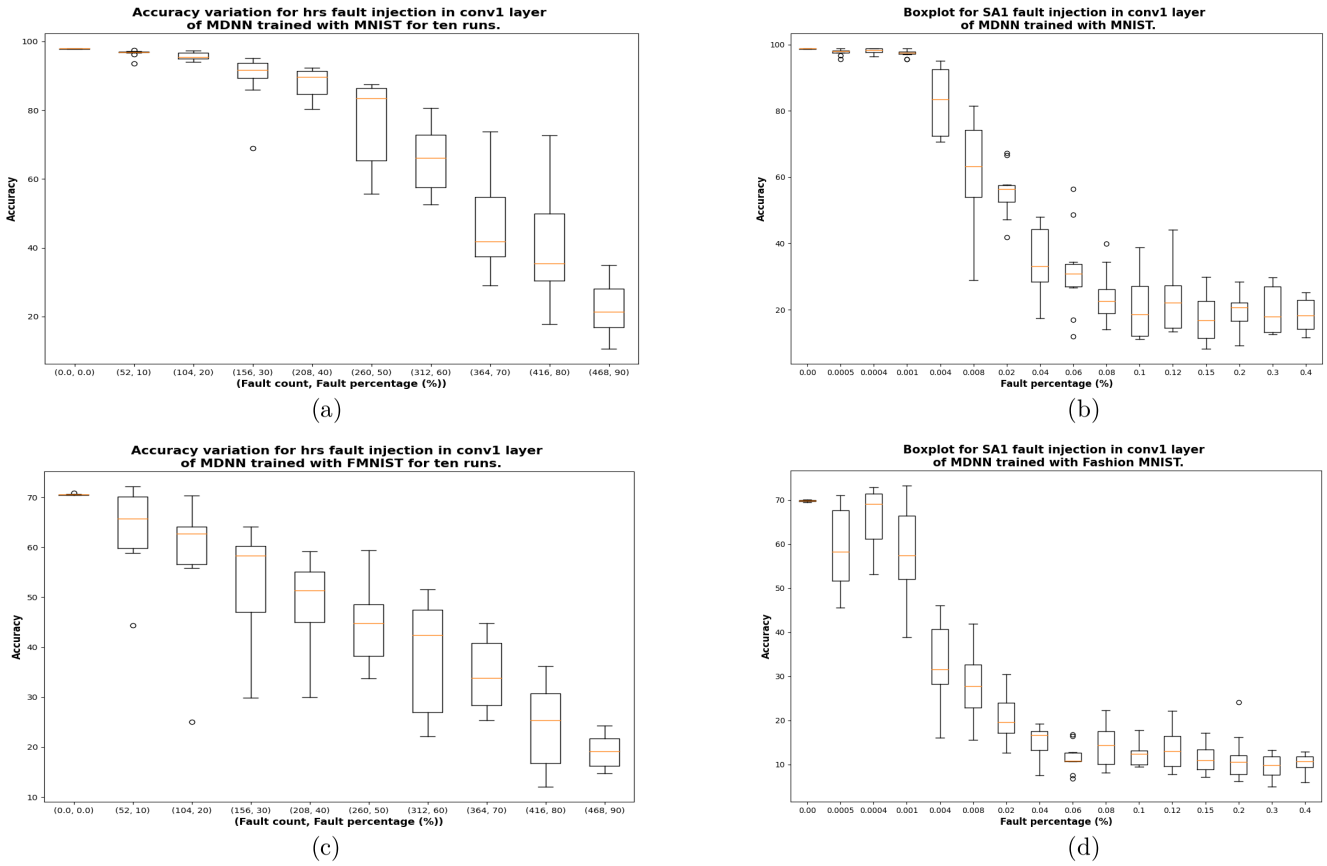


Figure 7.9: Accuracy range for ten simulations of (a) MDNN architecture trained for MNIST with $hrs\ SA0$ faults injected, (b) MDNN architecture trained for MNIST with $lrs\ SA1$ faults injected, (c) MDNN architecture trained for Fashion MNIST with $hrs\ SA0$ faults injected, and (d) MDNN architecture trained for Fashion MNIST with $lrs\ SA1$ faults injected.

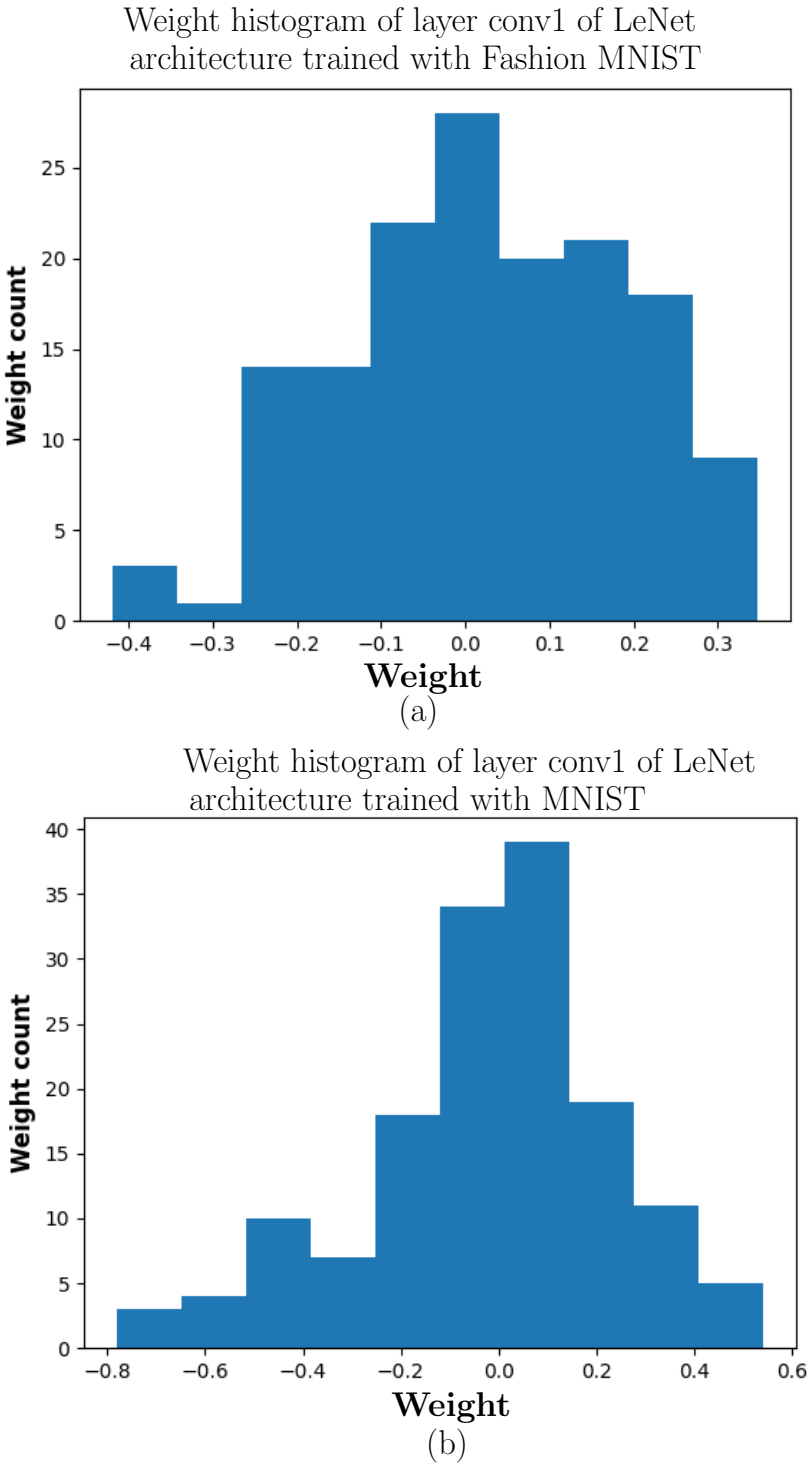


Figure 7.10: Histogram of weights in LeNet architecture trained with (a) Fashion MNIST dataset, and (b) MNIST dataset, respectively.

CHAPTER

EIGHT

CONCLUSIONS AND FUTURE DIRECTIONS

Contents

This thesis contributes to advancing memristor-based computing systems by addressing fault sensitization, testing optimization, ALU design, and the impact of faults on neuromorphic architectures. The findings provide valuable insights to improve the reliability and performance of future fault-tolerant memristor-based systems across a wide range of applications. The novel contributions and outline future plans in Sections 8.1 and 8.2. These sections provide a concise summary of the original findings and their implications, as well as the intended directions for further exploration and development in our research.

8.1 Summary

The thesis contributions are organized on a chapter-by-chapter basis as follows:

- In Chapter 5, we envisage a 2D memristive crossbar as a network and identify certain paths that are suitable for fault sensitization. For full-size square and rectangular memristive crossbars, the proposed method optimizes test time using a path-based

technique guided by maximum matching in bipartite graphs. Simulation results with LTspice demonstrate the effectiveness and superiority of the method to the prior art in terms of test time and fault-coverage.

- We observe that the graph-based path-covering techniques proposed in Chapter 5 reduces testing time efficiently for square or a rectangular full crossbar. But, this techniques may not always be applicable for crossbar with irregular structure or for an incomplete crossbar. In order to cover all types of crossbar (including incomplete), in Chapter 6 we have proposed an Mixed Integer Linear Program (MILP) formulation for optimal path covering that can uniformly handle both full and incomplete crossbars.
- In Chapter 3 we propose a logic system based on differential currents for implementing ADDER on a hybrid-memristor crossbar network. The addition is performed in the binary domain, using both analog and digital components. The analog component provides the peripheral control circuit, input voltages and logic values to the crossbar in the form of memristor-states. The variation of the analog output current is then sensed and converted back to discrete logic bits using A/D converters. Simulation studies demonstrate that the proposed design reduces both memristor cost and computation-cycle time compared to previous approaches.
- In Chapter 4, we propose designs for arithmetic and logical circuits on a memristor-based hybrid crossbar network that relies on current sensing along with some analog peripheral circuits. Thus, computation is performed in a mixed domain, i.e., a binary input is mapped as a state of a memristor in the crossbar topology, whereas the outcome of the arithmetic and logic operations appear in the analog domain, a hybridized strategy that leads to reduced computation time and improved energy efficiency.
- In chapter 7, we conducted a comprehensive analysis of the effect of stuck-at-zero ($SA0$) and stuck-at-one ($SA1$) faults on the accuracy degradation of an MNN consisting of two convolutional layers, two fully connected layers, and a memristive LeNet network. This area of research is particularly important because understanding the influence of such faults and variations can contribute to the development of fault-tolerant neuromorphic computing systems with limited resources. Additionally, by gaining insight into the impact of faults on individual layers of memristive crossbar-based neural architectures, we may discover optimal architecture designs for solving specific problems with predefined accuracy.

8.2 Future Directions

There are several promising directions for future research based on the key contributions of this thesis. By combining the proposed techniques, it is possible to develop fast and energy-efficient memristive crossbar-based fault-tolerant neuromorphic computing systems, even with limited resources.

One important future direction is to focus on the development of memristive crossbar-based fault models. Investigating the impact of different faults on circuit operation, performance, and system reliability can lead to significant advancements in the development of fault-tolerant memristive crossbar-based neuromorphic computing systems.

Additionally, there is great potential in designing energy-efficient operations for neuromorphic systems that utilize minimal peripheral components. By optimizing system architecture and reducing unnecessary overhead, it is possible to achieve higher energy efficiency while maintaining desired performance levels. Exploring novel techniques and approaches to achieve such energy-efficient operation holds promise for future advancements.

Another important future direction is the analysis and understanding of the effects of variability, non-ideal characteristics, and read-write noise on system performance. Gaining deeper insights into these factors and developing strategies to mitigate their impact can enhance the reliability and performance of memristor-based neuromorphic systems.

Overcoming these challenges will be crucial for the realization of efficient and high-performance memristor-based neuromorphic systems with transformative capabilities in cognitive computing and artificial intelligence. Continued research efforts in these future directions will pave the way for the practical implementation and widespread adoption of such advanced systems.

BIBLIOGRAPHY

[AFC15] A. G. Alharbi, M. E. Fouda, M. H. Chowdhury. Memristor emulator based on practical current controlled model. In *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, 2015. doi:10.1109/MWSCAS.2015.7282109.

[AHSP23] C. Aitchison, B. Halak, A. Serb, T. Prodromakis. A memristor fingerprinting and characterisation methodology for hardware security. *Scientific Reports*, 13(1):9392, June 2023. URL https://eprints.soton.ac.uk/477281/.

[AJB00] R. Albert, H. Jeong, A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, Jul 2000. doi:10.1038/35019019.

[Aka15] S. Akashe. Design and Analysis of Memristor Based Non-volatile Memories. In *Springer Proceedings in Physics*, volume 166 of *Springer Proceedings in Physics*, page 107, jan 2015. doi:10.1007/978-81-322-2367-2_14.

[AMY+23] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, J. S. Vetter. A survey on processing-in-memory techniques: Advances and challenges. *Memories - Materials, Devices, Circuits and Systems*, 4:100022, 2023. doi:https://doi.org/10.1016/j.memori.2022.100022.

[ARB+19] K. Ali, M. Rizk, A. Baghdadi, others. MRL crossbar-based full adder design. In *Proc. ICECS*, pages 674–677, 2019.

[ASKJ18] K. Adam, K. Smagulova, O. Krestinskaya, A. James. Wafer quality inspection using memristive lstm, ann, dnn and htm. In *2018 IEEE Electrical*

Design of Advanced Packaging and Systems Symposium (EDAPS), pages 1–3, 12 2018. doi:10.1109/EDAPS.2018.8680907.

[AT16] R. Aluguri, T.-Y. Tseng. Notice of violation of ieee publication principles: Overview of selector devices for 3-d stackable cross point rram arrays. *IEEE Journal of the Electron Devices Society*, 4(5):294–306, 2016. doi:10.1109/JEDS.2016.2594190.

[AWY+23] J. An, L. Wang, W. Ye, W. Li, H. Gao, Z. Li, Z. Zhou, J. Tian, J. Gao, C. Dou, Q. Liu. Design memristor-based computing-in-memory for AI accelerators considering the interplay between devices, circuits, and system. *Science China Information Sciences*, 66(8):182404, 2023.

[AYS+19] A. Awad, M. Ye, Y. Solihin, L. Njilla, K. A. Zubair. Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 104–115, 2019.

[BA00] M. L. Bushnell, V. D. Agarwal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits.* Springer, 2000.

[BA13] M. Bushnell, V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits.* Springer Publishing Company, Incorporated, 2013.

[BAC18] D. Bhattacharjee, L. Amaŕu, A. Chattopadhyay. Technology-aware logic synthesis for reram based in-memory computing. In *Proc. Design, Automation Test in Europ*, pages 1435–1440, 2018.

[BBB09] Z. Biolek, D. Biolek, V. Biolková. Spice model of memristor with nonlinear dopant drift. *Radioengineering*, pages 210–214, 2009.

[BBJP19] L. Batina, S. Bhasin, D. Jap, S. Picek. Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 515–532, USA, 2019. USENIX Association.

[BGS+18] I. Boybat, M. Gallo, N. S.R., T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, E. Eleftheriou. Neuromorphic computing with multi-memristive synapses. *Nature Communications*, 9, 06 2018. doi:10.1038/s41467-018-04933-y.

[BPC15] H. Bao, J. Park, J. Cao. Exponential synchronization of coupled stochastic memristor-based neural networks with time-varying probabilistic delay coupling and impulsive delay. *IEEE transactions on neural networks and learning systems*, 27, 10 2015. doi:10.1109/TNNLS.2015.2475737.

[BS10] K. Bickerstaff, E. E. Swartzlander. Memristor-based arithmetic. In *Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*, pages 1173–1177, 2010.

[BS14] K. Bickerstaff, E. E. Swartzlander. *Memristor-Based Addition and Multiplication*, pages 473–486. Springer International Publishing, 01 2014. doi:10.1007/978-3-319-02630-5_21.

[BS17] S. Brivio, S. Spiga. Stochastic circuit breaker network model for bipolar resistance switching memories. *J. Comput. Electron.*, 16(4):1154–1166, dec 2017. doi:10.1007/s10825-017-1055-y.

[BSK+10] J. Borghetti, G. Snider, P. Kuekes, J. Yang, D. Stewart, R. Williams. Memristive switches enable stateful logic operations via material implication. *Nature*, 464(873), 2010.

[BSS+17] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. L. Gallo, K. Moon, J. Woo, H. Hwang, Y. Leblebici. Neuromorphic computing using non-volatile memory. *Advances in Physics: X*, 2(1):89–124, 2017. doi:10.1080/23746149.2016.1259585.

[BTP13] J. Bürger, C. Teuscher, M. Perkowski. Digital logic synthesis for memristors. In *slidepalyer.com*, pages 31–40, 01 2013.

[BZL+22] H. Bao, H. Zhou, J. Li, H. Pei, others. Toward memristive in-memory computing: principles and applications. *Frontiers of Optoelectronics*, 15(2):23, 2022.

[CC20] L. Chen, Z. Chu. Towards optimal logic representations for implication-based memristive circuits. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pages 1–3, 2020. doi:10.1109/CSTIC49141.2020.9282401.

[CC21a] C.-Y. Chen, K. Chakrabarty. Efficient identification of critical faults in memristor crossbars for deep neural networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1074–1077, 2021. doi:10.23919/DATE51398.2021.9473989.

[CC21b] C.-Y. Chen, K. Chakrabarty. Pruning of deep neural networks for fault-tolerant memristor-based accelerators. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 889–894, 2021.

[CC22] C.-Y. Chen, K. Chakrabarty. Efficient identification of critical faults in memristor-based inferencing accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(7):2301–2314, 2022. doi:10.1109/TCAD.2021.3102894.

[CCSZ23] B. Coqueret, M. Carbone, O. Sentieys, G. Zaid. When side-channel attacks break the black-box property of embedded artificial intelligence. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, page 127–138, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3605764.3623903.

[CCT+19] Y. Cai, X. Chen, L. Tian, Y. Wang, H. Yang. Enabling secure in-memory neural network computing by sparse fast gradient encryption. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019. doi:10.1109/ICCAD45719.2019.8942041.

[Chu71] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971.

[CJ17a] D. Chakraborty, S. K. Jha. Automated synthesis of compact crossbars for sneak-path based in-memory computing. In *Proc. Design, Automation Test in Europe*, pages 770–775, 2017.

[CJ17b] D. Chakraborty, S. K. Jha. Automated synthesis of compact crossbars for sneak-path based in-memory computing. In *Design, Automation & Test*

in Europe Conference & Exhibition (DATE), 2017, pages 770–775, 2017. doi:10.23919/DATE.2017.7927093.

[CKS+10] B. Cho, T.-W. Kim, S. Song, Y. Ji, M. Jo, H. Hwang, G.-Y. Jung, T. Lee. Rewritable switching of one diode–one resistor nonvolatile organic memory devices. *Advanced Materials*, 22(11):1228–1232, 2010.

[CL12] W. Chan, J. Lohn. Spike timing dependent plasticity with memristive synapse in neuromorphic systems. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2012. doi:10.1109/IJCNN.2012.6252822.

[CL15] Y.-X. Chen, J.-F. Li. Fault modeling and testing of 1t1r memristor memories. In *2015 IEEE 33rd VLSI Test Symposium (VTS)*, pages 1–6, 2015. doi:10.1109/VTS.2015.7116247.

[CLC19] A. Chaudhuri, M. Liu, K. Chakrabarty. Fault-tolerant neuromorphic computing systems. In *2019 IEEE International Test Conference (ITC)*, pages 1–10, 2019. doi:10.1109/ITC44170.2019.9000146.

[CSAE19] L. Chua, G. C. Sirakoulis, A. Adamatzky (Eds.). *Handbook of Memristor Networks*. Springer Nature, Switzerland AG, 2019.

[CSW+15] C. Y. Chen, H. C. Shih, C. W. Wu, C. H. Lin, P. F. Chiu, S. S. Sheu, F. T. Chen. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Transactions on Computers*, 64(1):180–190, 2015.

[CTSC20] A. Chaudhuri, J. Talukdar, F. Su, K. Chakrabarty. Functional criticality classification of structural faults in ai accelerators. In *Proc. International Test Conference*, pages 1–5, 2020.

[CZQK11] D. Chabi, W. Zhao, D. Querlioz, J.-O. Klein. Robust neural logic block (nlb) based on memristor crossbar array. In *2011 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 137–143, 2011. doi:10.1109/NANOARCH.2011.5941495.

[DBSW08] D. R. S. Dmitri B. Strukov, R. S. Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[Den12] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[DFR+15] R. Degraeve, A. Fantini, N. Raghavan, L. Goux, S. Clima, B. Govoreanu, A. Belmonte, D. Linten, M. Jurczak. Causes and consequences of the stochastic aspect of filamentary rram. *Microelectron. Eng.*, 147:171–175, nov 2015.

[DK20] N. C. Dao, D. Koch. Memristor-based reconfigurable circuits: Challenges in implementation. In *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–6, 2020. doi:10.1109/ICEIC49074.2020.9051174.

[EFR15] A. A. El-Slehdar, A. H. Fouad, A. G. Radwan. Memristor based n-bits redundant binary adder. *Microelectronics Journal*, 46(3):207–213, 2015.

[EVR19] M. Escudero, I. Vourkas, A. Rubio. Stuck-at-off fault analysis in memristor-based architecture for synchronization. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 33–37, 2019. doi:10.1109/IOLTS.2019.8854413.

[FSD19] S. Fröhlich, S. Shirinzadeh, R. Drechsler. Logic synthesis for hybrid cmos-reram sequential circuits. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 431–436, 2019. doi:10.1109/ISVLSI.2019.00084.

[GKR19] P. Ganesh, S. Krishna, V. Ravi. Multi-level memristor memory: Design and performance analysis. *International Journal of Innovative Technology and Exploring Engineering*, 8:723–729, 01 2019.

[GPJ+22] T. Guo, K. Pan, Y. Jiao, B. Sun, C. Du, J. P. Mills, Z. Chen, X. Zhao, L. Wei, Y. N. Zhou, Y. A. Wu. Versatile memristor for memory and neuromorphic computing. *Nanoscale Horiz.*, 7:299–310, 2022. doi:10.1039/D1NH00481F.

[GPM09a] O. Ginez, J. M. Portal, C. Muller. Design and test challenges in resistive switching RAM (ReRAM): An electrical model for defect injections. In *Proc. European Test Symposium*, pages 61–66, 2009.

[GPM09b] O. Ginez, J.-M. Portal, C. Muller. Design and test challenges in resistive switching ram (reram): An electrical model for defect injections. In *2009 14th IEEE European Test Symposium*, pages 61–66, 2009. doi:10.1109/ETS.2009.23.

[GS17a] L. Guckert, E. E. Swartzlander. Optimized memristor-based multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(2):373–385, 2017.

[GS17b] L. Guckert, E. E. Swartzlander. Dadda multiplier designs using memristors. In *Proc. ICICDT*, pages 1–4, 2017.

[HAMC19] B. Hajri, H. Aziza, M. Mansour, A. Chehab. Rram device models: A comparative analysis with experimental validation. *IEEE Access*, PP:1–1, 11 2019. doi:10.1109/ACCESS.2019.2954753.

[HAS14] S. Hamdioui, H. Aziza, G. C. Sirakoulis. Memristor based memories: Technology, design and test. In *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–7, 2014. doi:10.1109/DTIS.2014.6850647.

[HCJ18] A. U. Hassen, D. Chakraborty, S. K. Jha. Free binary decision diagram-based synthesis of compact crossbars for in-memory computing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(5):622–626, 2018.

[HFNT19] S. Hamdioui, M. Fieback, S. Nagarajan, M. Taouil. Testing computation-in-memory architectures based on emerging memories. In *2019 IEEE International Test Conference (ITC)*, pages 1–10, 2019. doi:10.1109/ITC44170.2019.9000117.

[HGL+18a] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, J. P. Strachan. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018. doi:https://doi.org/10.1002/adma.201705914.

[HGL+18b] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, J. P. Strachan. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018. doi:https://doi.org/10.1002/adma.201705914.

[HH11] N. Z. Haron, S. Hamdioui. On defect oriented testing for hybrid cmos/memristor memory. In *Proc. Asian Test Symposium*, pages 353–358, 2011.

[HHL09] Y. Ho, G. M. Huang, P. Li. Nonvolatile memristor memory: Device characteristics and design implications. In *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pages 485–490, 2009.

[HKKC11] V. A. Hongal, R. Kotikalapudi, Y. B. Kim, M. Choi. A novel " divide and conquer " testing technique for memristor based lookup table. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, 2011. doi:10.1109/MWSCAS.2011.6026406.

[HPJ+20] S. Huang, X. Peng, H. Jiang, Y. Luo, S. Yu. New security challenges on machine learning inference engine: Chip cloning and model reverse engineering, 2020, 2003.09739.

[HTH15] S. Hamdioui, M. Taouil, N. Z. Haron. Testing open defects in memristor-based memories. *IEEE Transactions on Computers*, 64(1):247–259, 2015.

[HTY17] R. Hasan, T. M. Taha, C. Yakopcic. On-chip training of memristor crossbar based multi-layer neural networks. *Microelectronics Journal*, 66:31–40, 2017. doi:https://doi.org/10.1016/j.mejo.2017.05.005.

[HXZ+21] W. Huang, X. Xia, C. Zhu, P. Steichen, W. Quan, W. Mao, J. Yang, L. Chu, X. Li. Memristive artificial synapses for neuromorphic computing. *Nano-Micro Letters*, 13:1–28, 2021.

[HZS18] W. Hua, Z. Zhang, G. E. Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018. doi:10.1109/DAC.2018.8465773.

[JA22] R. Joshi, J. M. Acken. Fault coverage analysis using sneak path based testing in memristor circuits. In *2022 IEEE 31st Microelectronics Design & Test Symposium (MDTS)*, pages 1–6, 2022. doi:10.1109/MDTS54894.2022.9826959.

[JLZ+18] H. Jiang, C. Li, R. Zhang, P. Yan, P. Lin, Y. Li, J. J. Yang, D. Holcomb, Q. Xia. A provable key destruction scheme based on memristive crossbar arrays. *Nature Electronics*, 1(10):548–554, 2018.

[JW08] Y. Joglekar, S. Wolf. The elusive memristor: properties of basic electrical circuits. *European Journal of Physics*, 30, 07 2008. doi:10.1088/0143-0807/30/4/001.

[KASC+11] O. Kavehei, S. Al-Sarawi, K. R. Cho, N. Iannella, S. J. Kim, K. Eshraghian, D. Abbott. Memristor-based synaptic networks and logical operations using in-situ computing. In *2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 137–142, 2011. doi:10.1109/ISSNIP.2011.6146610.

[KBL+14] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. Friedman, A. Kolodny, U. Weiser. Magic—memristor-aided logic. *TCAS II: Express Briefs*, 61(11):895–899, 2014.

[KD15] A. D. Keedwell, J. Dénes. *Latin Squares and Their Applications*. Elsevier, 2015.

[KDRB16] M. Kule, A. Dutta, H. Rahaman, B. B. Bhattacharya. High-speed decoder design using memristor-based nano-crossbar architecture. In *Proc. Sixth International Symposium on Embedded Computing and System Design*, pages 77–81, Dec 2016.

[KFKW13] S. Kvatinsky, E. G. Friedman, A. Kolodny, U. C. Weiser. Team: Threshold adaptive memristor model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):211–221, 2013. doi:10.1109/TCSI.2012.2215714.

[KJC20] O. Krestinskaya, A. P. James, L. O. Chua. Neuromemristive circuits for edge computing: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 31(1):4–23, 2020. doi:10.1109/TNNLS.2019.2899262.

[KKKS14] S. Kannan, N. Karimi, R. Karri, O. Sinanoglu. Detection, diagnosis, and repair of faults in memristor-based memories. In *2014 IEEE 32nd VLSI Test Symposium (VTS)*, pages 1–6, 2014. doi:10.1109/VTS.2014.6818762.

[KKKS15] S. Kannan, N. Karimi, R. Karri, O. Sinanoglu. Modeling, detection, and diagnosis of faults in multilevel memristor memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 822–834, 2015.

[KMNS21] H. Kim, M. R. Mahmoodi, H. Nili, D. B. Strukov. 4k-memristor analog-grade passive crossbar circuit. *Nature Communications*, 12(1):5198, 2021.

[KP23] K. Kishori, S. Pyne. In-memory set operations on memristor crossbar. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2023. doi:10.1109/TCAD.2023.3294450.

[KRFK15] S. Kvatinsky, M. Ramadan, E. G. Friedman, A. Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015. doi:10.1109/TCSII.2015.2433536.

[KRKS13a] S. Kannan, J. Rajendran, R. Karri, O. Sinanoglu. Sneak-path testing of crossbar-based nonvolatile random access memories. *IEEE Transactions on Nanotechnology*, pages 413–426, 2013.

[KRKS13b] S. Kannan, J. Rajendran, R. Karri, O. Sinanoglu. Sneak-path testing of memristor-based memories. In *Proc. 26th International Conference on VLSI Design and Embedded Systems*, pages 386–391, 2013.

[KS11] M. Klimo, O. Such. Memristors can implement fuzzy logic. *CoRR*, abs/1110.2074, 2011, 1110.2074. URL http://arxiv.org/abs/1110.2074.

[KSW+14] S. Kvatinsky, G. Satat, N. Wald, E. Friedman, A. Kolodny, U. C. Weiser. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. *IEEE Transactions on Very Large Scale Integration Systems*, 22:2054–2066, 2014.

[KUH19] S. Khokhar, A. Ul Hassen. Synthesis of approximate logic on memristive crossbars. In *2019 17th IEEE International New Circuits and Systems*

Conference (NEWCAS), pages 1–4, 2019. doi:10.1109/NEWCAS44328.2019.8961297.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[LBJ+17] T. Li, X. Bi, N. Jing, X. Liang, L. Jiang. Sneak-path based test and diagnosis for 1r rram crossbar using voltage bias technique. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. doi:10.1145/3061639.3062318.

[LBL+18] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia. In-memory computing with memristor arrays. In *2018 IEEE International Memory Workshop (IMW)*, pages 1–4, 2018. doi:10.1109/IMW.2018.8388838.

[LBSGCM+11] B. Linares-Barranco, T. Serrano-Gotarredona, L. Camuñas-Mesa, J. Perez-Carrasco, C. Zamarreño-Ramos, T. Masquelier. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in Neuroscience*, 5, 2011. doi:10.3389/fnins.2011.00026.

[LCLL21] N. Lin, X. Chen, H. Lu, X. Li. Chaotic weights: A novel approach to protect intellectual property of deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1327–1339, 2021. doi:10.1109/TCAD.2020.3018403.

[LCX+22] C. Liu, C. Chu, D. Xu, Y. Wang, Q. Wang, H. Li, X. Li, K.-T. Cheng. Hyca: A hybrid computing architecture for fault-tolerant deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3400–3413, 2022. doi:10.1109/TCAD.2021.3124763.

[LHSL17] C. Liu, M. Hu, J. P. Strachan, H. H. Li. Rescuing memristor-based neuromorphic design with high defects. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3061639.3062310.

[LLY+14] H. Liu, H. Lv, B. Yang, X. Xu, R. Liu, Q. Liu, S. Long, M. Liu. Uniformity improvement in 1t1r rram with gate voltage ramp programming. *IEEE Electron Device Letters*, 35(12):1224–1226, 2014. doi:10.1109/LED.2014.2364171.

[LNM05] R. Legenstein, C. Naeger, W. Maass. What can a neuron learn with spike-timing-dependent plasticity? *Neural computation*, 17:2337–82, 12 2005. doi:10.1162/0899766054796888.

[LPL+21a] A. Lu, X. Peng, W. Li, H. Jiang, S. Yu. Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark. *Frontiers in Artificial Intelligence*, 4, 2021. URL https://api.semanticscholar.org/CorpusID:235373135.

[LPL+21b] A. Lu, X. Peng, W. Li, H. Jiang, S. Yu. Neurosim validation with 40nm rram compute-in-memory macro. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4, 2021. doi:10.1109/AICAS51828.2021.9458501.

[LWJ+19] T. Liu, W. Wen, L. Jiang, Y. Wang, C. Yang, G. Quan. A fault-tolerant neural network architecture. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[LWY+18] P. Liu, J. Wu, Z. You, M. Elimu, W. Wang, S. Cai. Defect analysis and parallel march test algorithm for 3d hybrid cmos-memristor memory. In *2018 IEEE 27th Asian Test Symposium (ATS)*, pages 25–29, 2018. doi:10.1109/ATS.2018.00016.

[LXLBR22] C. Lammie, W. Xiang, B. Linares-Barranco, M. Rahimi Azghadi. Memtorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing*, 485:124–133, 2022. doi:https://doi.org/10.1016/j.neucom.2022.02.043.

[LXY+17] H. Lv, X. Xu, P. Yuan, D. Dong, T. Gong, J. Liu, Z. Yu, P. Huang, K. Zhang, C. Huo, C. Chen, Y. Xie, Q. Luo, S. Long, Q. Liu, J. Kang, D. Yang, S. Yin, S. Chiu, M. Liu. Beol based rram with one extra-mask for low cost, highly reliable embedded application in 28 nm node and be-

yond. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 2.4.1–2.4.4, 2017. doi:10.1109/IEDM.2017.8268312.

[LZW20] X. Liu, Z. Zeng, D. C. Wunsch II. Memristor-based lstm network with in situ training and its applications. *Neural Networks*, 131:300–311, 2020.

[Man79] M. M. Mano. *Digital Logic and Computer Design*. Prentice Hall PTR, USA, 1st edition, 1979.

[MRHW10] H. Manem, G. S. Rose, X. He, W. Wang. Design considerations for variation tolerant multilevel cmos/nano memristor memory. In *Proc. Symposium on Great Lakes Symposium on VLSI*, page 287–292, 2010.

[MRR12] H. Manem, J. Rajendran, G. S. Rose. Stochastic gradient descent inspired training technique for a cmos/nano memristive trainable threshold gate array. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(5):1051–1060, 2012. doi:10.1109/TCSI.2012.2190665.

[MSB19] M. Mondal, S. Sur-Kolay, B. B. Bhattacharya. Selective sensitization of useless sneak-paths for test optimization in memristor-arrays. In *Proc. 32nd International Conference on VLSID*, pages 383–388, 2019.

[MSR+20] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, A. J. Kenyon. Memristors—from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing. *Advanced Intelligent Systems*, 2(11):2000085, 2020, https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202000085. doi:https://doi.org/10.1002/aisy.202000085.

[MTH17] S. N. Mozaffari, S. Tragoudas, T. Haniotakis. More efficient testing of metal-oxide memristor–based memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(6):1018–1029, 2017. doi:10.1109/TCAD.2016.2608863.

[NOBT12] A. Nere, U. Olcese, D. Balduzzi, G. Tononi. A neuromorphic architecture for object recognition and motion anticipation using burst-stdp. *PLOS ONE*, 7(5):1–17, 05 2012. doi:10.1371/journal.pone.0036958.

[PAR15] P. Pouyan, E. Amat, A. Rubio. Memristive crossbar design and test in non-adaptive proactive reconfiguring scheme. In *2015 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, 2015. doi:10.1109/ECCTD.2015.7300125.

[PFHE+21] L. Poehls, M. Fieback, S. Hoffmann-Eifert, S. C. T., E. Brum, S. Menzel, S. Hamdioui, T. Gemmeke. Review of manufacturing process defects and their effects on memristive devices. *Journal of Electronic Testing*, 37:1–11, 08 2021. doi:10.1007/s10836-021-05968-8.

[Piu01] V. Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61:18–48, 01 2001. doi:10.1006/jpdc.2000.1663.

[PMBH+15] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. Likharev, D. Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Microelectron. Eng.*, 521(7550):61–64, nov 2015.

[PW20] P. Pal, Y. Wang. Interconversion of complementary resistive switching from graphene oxide based bipolar multilevel resistive switching device. *Applied Physics Letters*, 117(5), Aug. 2020. doi:10.1063/5.0010319.

[PZP21] F. M. Puglisi, T. Zanotti, P. Pavan. Optimized synthesis method for ultra-low power multi-input material implication logic with emerging non-volatile memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(11):4433–4443, 2021. doi:10.1109/TCSI.2021.3079986.

[QBDG13] D. Querlioz, O. Bichler, P. Dollfus, C. Gamrat. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology*, 12:288–295, 05 2013. doi:10.1109/TNANO.2013.2250995.

[QBG11] D. Querlioz, O. Bichler, C. Gamrat. Simulation of a memristor-based spiking neural network immune to device variations. In *The 2011 International Joint Conference on Neural Networks*, pages 1775–1781, 2011. doi:10.1109/IJCNN.2011.6033439.

[QPMW21] E. P.-B. Quesada, E. Perez, M. K. Mahadevaiah, C. Wenger. Memristive-based in-memory computing: from device to large-scale cmos integration. *Neuromorphic Computing and Engineering*, 1(2):024006, nov 2021. doi:10.1088/2634-4386/ac2cd4.

[REL+22] R. Ronen, A. Eliahu, O. Leitersdorf, N. Peled, K. Korgaonkar, A. Chattopadhyay, B. Perach, S. Kvatinsky. The bitlet model: A parameterized analytical model to compare pim and cpu systems. *J. Emerg. Technol. Comput. Syst.*, 18(2), mar 2022. doi:10.1145/3465371.

[RK21] M. Raj, G. Kavithaa. Memristor based high speed and low power consumption memory design using deep search method. *Journal of Ambient Intelligence and Humanized Computing*, 12, 03 2021. doi:10.1007/s12652-020-01817-2.

[RRP+23] D. Rajasekharan, N. Rangarajan, S. Patnaik, O. Sinanoglu, Y. S. Chauhan. Scanet: Securing the weights with superparamagnetic-mtj crossbar array networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(9):5693–5707, 2023. doi:10.1109/TNNLS.2021.3130884.

[RS18] N. Revanna, E. E. Swartzlander. Memristor adder design. In *Proc. MWSCAS*, pages 314–317, 2018.

[SBK+22] S. H. H. Shadmehri, A. BanaGozar, M. Kamal, S. Stuijk, A. Afzali-Kusha, M. Pedram, H. Corporaal. Syscim: Systemc-ams simulation of memristive computation in-memory. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1467–1472, 2022. doi:10.23919/DATE54114.2022.9774749.

[SFP+23] F. Staudigl, T. Fetz, R. Pelke, D. Sisejkovic, J. M. Joseph, L. B. Pöhls, R. Leupers. Fault injection in native logic-in-memory computation on neuromorphic hardware, 2023, 2302.07655.

[SGMP+13] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, B. Linares-Barranco. Stdp and stdp variations with memristors for spiking neuromorphic learning systems. *Frontiers in Neuroscience*, 7, 2013. doi:10.3389/fnins.2013.00002.

[SHY18] C. Sung, H. Hwang, I. K. Yoo. Perspective: A review on memristive hardware for neuromorphic computation. *Journal of Applied Physics*, 124(15):151903, 10 2018. doi:10.1063/1.5037835.

[SKM+13] M. Soltiz, D. Kudithipudi, C. Merkel, G. S. Rose, R. E. Pino. Memristor-based neural logic blocks for nonlinearly separable functions. *IEEE Transactions on Computers*, 62(8):1597–1606, 2013. doi:10.1109/TC.2013.75.

[SL12] D. B. Strukov, K. K. Likharev. Reconfigurable nano-crossbararchitectures. *Nanoelectronics and Information Technology*, page 435–454, 2012.

[SMK+19] S. Stathopoulos, L. Michalas, A. Khiat, A. Serb, T. Prodromakis. An electrical characterisation methodology for benchmarking memristive device technologies. *Scientific Reports*, 9(1):19412, dec 2019.

[SML21] F. Staudigl, F. Merchant, R. Leupers. A survey of neuromorphic computing-in-memory: Architectures, simulators and security. *IEEE Design Test*, pages 1–1, 2021.

[SSB+22] F. Staudigl, K. J. X. Sturm, M. Bartel, T. Fetz, D. Sisejkovic, J. M. Joseph, L. B. Pöhls, R. Leupers. X-fault: Impact of faults on binary neural networks in memristor-crossbar arrays with logic-in-memory computation. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 174–177, 2022. doi:10.1109/AICAS54282.2022.9869897.

[SSGD16] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, R. Drechsler. Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 948–953, 2016.

[SSS+20] F. Silva, M. Sanz, J. Seixas, E. Solano, Y. Omar. Perceptrons from memristors. *Neural Networks*, 122:273–278, 2020. doi:https://doi.org/10.1016/j.neunet.2019.10.013.

[SZT+20] L. Shi, G. Zheng, B. Tian, B. Dkhil, C. Duan. Research progress on solutions to the sneak path issue in memristor crossbar arrays. *Nanoscale Adv.*, 2:1811–1827, 2020. doi:10.1039/D0NA00100G.

[SZZM18a] L. Sun, N. Zheng, T. Zhang, P. Mazumder. Fault modeling and parallel testing for 1T1M memory array. *IEEE Transactions on Nanotechnology*, 17(3):437–451, 2018.

[SZZM18b] L. Sun, N. Zheng, T. Zhang, P. Mazumder. Fault modeling and parallel testing for 1t1m memory array. *IEEE Transactions on Nanotechnology*, 17(3):437–451, 2018. doi:10.1109/TNANO.2018.2806938.

[TAAA16] M. Teimoori, A. Ahmadi, S. Alirezaee, M. Ahmadi. A novel hybrid cmos-memristor logic circuit using memristor ratioed logic. In *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, 2016. doi:10.1109/CCECE.2016.7726661.

[TAS+14] M. Teimoory, A. Amirsoleimani, J. Shamsi, A. Ahmadi, S. Alirezaee, M. Ahmadi. Optimized implementation of memristor-based full adder by material implication logic. In *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 562–565, 2014. doi:10.1109/ICECS.2014.7050047.

[TGY+17] P. L. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, I. Sengupta. Efficient implementation of adder circuits in memristive crossbar array. In *Proc. TENCON 2017*, pages 207–212, 2017.

[THG17] C. Torres-Huitzil, B. Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 5:17322–17341, 2017.

[TRB+19] V. Tenace, R. G. Rizzo, D. Bhattacharjee, A. Chattopadhyay, A. Calimera. Said: A supergate-aided logic synthesis flow for memristive crossbars. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 372–377, 2019. doi:10.23919/DATE.2019.8714939.

[vN93] J. von Neumann. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

[VPC09] M. D. Ventra, Y. V. Pershin, L. O. Chua. Circuit elements with memory: Memristors, memcapacitors, and meminductors. *Proceedings of the IEEE*, 97(10):1717–1724, 2009.

[VRK+09] P. Vontobel, W. Robinett, P. Kuekes, D. Stewart, J. Straznicky, S. Williams. Writing to and reading from a nano-scale crossbar memory based on memristors. *Nanotechnology*, 20:425204, 10 2009. doi:10.1088/0957-4484/20/42/425204.

[WCGW20] Y. Wang, Y. Cao, Z. Guo, S. Wen. Passivity and passification of memristive recurrent neural networks with multi-proportional delays and impulse. *Appl. Math. Comput.*, 369(C), mar 2020. doi:10.1016/j.amc.2019.124838.

[WLW+18] H. Wang, C. Lin, C. Wu, Y. Chen, C. Wang. On synthesizing memristor-based logic circuits with minimal operational pulses. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2842–2852, 2018.

[WRM+18] Z. Wang, M. Rao, R. Midya, S. Joshi, H. Jiang, P. Lin, W. Song, S. Asapu, Y. Zhuo, C. Li, H. Wu, Q. Xia, J. J. Yang. Threshold switching of ag or cu in dielectrics: Materials, mechanism, and applications. *Advanced Functional Materials*, 28(6):1704862, 2018. doi:https://doi.org/10.1002/adfm.201704862.

[WSYZ15] L. Wang, Y. Shen, Q. Yin, G. Zhang. Adaptive synchronization of memristor-based neural networks with time-varying delays. *IEEE Transactions on Neural Networks and Learning Systems*, 26:2033–2042, 09 2015. doi:10.1109/TNNLS.2014.2361776.

[WTL+10] C.-H. Wang, Y.-H. Tsai, K.-C. Lin, M.-F. Chang, Y.-C. King, C.-J. Lin, S.-S. Sheu, Y.-S. Chen, H.-Y. Lee, F. T. Chen, M.-J. Tsai. Three-dimensional 4f2 reram cell with cmos logic compatible process. In *2010 International Electron Devices Meeting*, pages 29.6.1–29.6.4, 2010. doi:10.1109/IEDM.2010.5703446.

[WWY+21] S. Wen, H. Wei, Y. Yang, Z. Guo, Z. Zeng, T. Huang, Y. Chen. Memristive lstm network for sentiment analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(3):1794–1804, 2021. doi:10.1109/TSMC.2019.2906098.

[WWZH18] S. Wen, H. Wei, Z. Zeng, T. Huang. Memristive fully convolutional network: An accurate hardware image-segmentor in deep learning. *IEEE Trans-*

actions on Emerging Topics in Computational Intelligence, 2(5):324–334, 2018. doi:10.1109/TETCI.2018.2829911.

[WXY+20] J. Wang, Q. Xu, B. Yuan, S. Chen, B. Yu, F. Wu. Reliability-driven neural network training for memristive crossbar-based neuromorphic computing systems. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2020. doi:10.1109/ISCAS45731.2020.9180923.

[XA19] Z. Xu, J. Abraham. Safety design of a convolutional neural network accelerator with error localization and correction. In *2019 IEEE International Test Conference (ITC)*, pages 1–10, 2019. doi:10.1109/ITC44170.2019.9000149.

[XCG+20] Q. Xu, S. Chen, H. Geng, B. Yuan, B. Yu, F. Wu, Z. Huang. Fault tolerance in memristive crossbar-based neuromorphic computing systems. *Integration*, 70:70–79, 2020.

[XDJX11] C. Xu, X. Dong, N. P. Jouppi, Y. Xie. Design implications of memristor-based rram cross-point structures. In *2011 Design, Automation & Test in Europe*, pages 1–6, 2011. doi:10.1109/DATE.2011.5763125.

[XLN+17] L. Xia, M. Liu, X. Ning, K. Chakrabarty, Y. Wang. Fault-tolerant training with on-line fault detection for rram-based neural computing systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. doi:10.1145/3061639.3062248.

[XNT+18] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, K. Bertels. A mapping methodology of boolean logic circuits on memristor crossbar. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):311–323, 2018.

[XRV17] H. Xiao, K. Rasul, R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017, 1708.07747.

[XWG+21] Q. Xu, J. Wang, H. Geng, S. Chen, X. Wen. Reliability-driven neuromorphic computing systems design. In *Design, Automation Test in Europe Conference Exhibition*, pages 1586–1591, 2021.

[XWZH18] X. Xie, S. Wen, Z. Zeng, T. Huang. Memristor-based circuit implementation of pulse-coupled neural network with dynamical threshold generators. *Neurocomputing*, 284:10–16, 2018. doi:https://doi.org/10.1016/j.neucom.2018.01.024.

[YAT17] C. Yakopcic, M. Z. Alom, T. M. Taha. Extremely parallel memristor crossbar architecture for convolutional neural network implementation. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1696–1703, 2017. doi:10.1109/IJCNN.2017.7966055.

[YBT+22] S. Yu, T. Bunnam, S. Triamlumlerd, M. Pracha, F. Xia, R. Shafik, A. Yakovlev. Energy-efficient neural network design using memristive mac unit. *Frontiers in Electronics*, 3, 2022. doi:10.3389/felec.2022.877629.

[YFT20a] M. Yan, C. W. Fletcher, J. Torrellas. Cache telepathy: Leveraging shared resource attacks to learn dnn architectures. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association.

[YFT20b] M. Yan, C. W. Fletcher, J. Torrellas. Cache telepathy: Leveraging shared resource attacks to learn dnn architectures. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association.

[YHZ+19] T. You, K. Huang, X. Zhao, A. Yi, C. Chen, W. Ren, T. Jin, J. Lin, Y. Shuai, W. Luo, M. Zhou, W. Yu, X. Ou. Engineering of self-rectifying filamentary resistive switching in linbo3 single crystalline thin film via strain doping. *Scientific Reports*, 9:19134, 12 2019. doi:10.1038/s41598-019-55628-3.

[YLL+20] C. Yang, B. Liu, H. Li, Y. Chen, M. Barnell, Q. Wu, W. Wen, J. Rajendran. Thwarting replication attack against memristor-based neuromorphic computing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2192–2205, 2020. doi:10.1109/TCAD.2019.2937817.

[YNA+20] J. Yu, R. Nane, I. Ashraf, M. Taouil, S. Hamdioui, H. Corporaal, K. Bertels. Skeleton-based synthesis flow for computation-in-memory architec-

tures. *IEEE Transactions on Emerging Topics in Computing*, 8(2):545–558, 2020. doi:10.1109/TETC.2017.2760927.

[YNX+18] J. Yu, H. A. D. Nguyen, L. Xie, M. Taouil, S. Hamdioui. Memristive devices for computation-in-memory. In *Proc. Design, Automation Test in Europe*, pages 1646–1651, 2018.

[YT18] D. N. Yadav, P. L. Thangkhiew. Towards an in-memory reconfiguration of arithmetic logical unit using memristor crossbar array. In *IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6, 2018.

[YTS+11] C. Yakopcic, T. Taha, G. Subramanyam, R. Pino, S. Rogers. A memristor device model. *Electron Device Letters, IEEE*, 32:1436 – 1438, 11 2011. doi:10.1109/LED.2011.2163292.

[YZS19] L. Yang, Z. Zeng, X. Shi. A memristor-based neural network circuit with synchronous weight adjustment. *Neurocomputing*, 363:114–124, 2019. doi:https://doi.org/10.1016/j.neucom.2019.06.048.

[ZCX+16] Q. Zhang, X. Cui, X. Xu, X. Wang, Z. Ma, S. Zhou. Sneak-path based test for 3d stacked one-transistor-n-rram array. In *2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 226–229, 2016. doi:10.1109/EDSSC.2016.7785249.

[ZHX19] P. Zuo, Y. Hua, Y. Xie. Supermem: Enabling application-transparent secure persistent memory with low overheads. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 479–492, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3352460.3358290.

[ZJL+18] M. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. Kushner, W. Lu. A general memristor-based partial differential equation solver. *Nature Electronics*, 1:411–420, 07 2018. doi:10.1038/s41928-018-0100-6.

[ZLB+22] M. Zahedi, M. A. Lebdeh, C. Bengel, D. Wouters, S. Menzel, M. Le Gallo, A. Sebastian, S. Wong, S. Hamdioui. Mnemosene: Tile architecture and simulator for memristor-based computation-in-memory. *J. Emerg. Technol. Comput. Syst.*, 18(3), jan 2022. doi:10.1145/3485824.

[ZM17] N. Zheng, P. Mazumder. Hardware-friendly actor-critic reinforcement learning through modulation of spike-timing-dependent plasticity. *IEEE Trans. on Computers*, 66(2):299–311, 2017.

[ZRCMPC+11] C. Zamarreño-Ramos, L. Camuñas-Mesa, J. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, B. Linares-Barranco. On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in neuroscience*, 5:26, 03 2011. doi:10.3389/fnins.2011.00026.

[ZUFE20] B. Zhang, N. Uysal, D. Fan, R. Ewetz. Handling stuck-at-fault defects using matrix transformation for robust inference of dnns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2448–2460, 2020. doi:10.1109/TCAD.2019.2944582.

[ZZC+22] M. Zou, J. Zhou, X. Cui, W. Wang, S. Kvatinsky. Enhancing security of memristor computing system through secure weight mapping. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 182–187, 2022. doi:10.1109/ISVLSI54635.2022.00044.