2018

In-Memory Computing Using Formal Methods and Paths-Based Logic

Alvaro Velasquez
University of Central Florida

IN-MEMORY COMPUTING USING FORMAL METHODS AND PATHS-BASED LOGIC

by

ALVARO VELASQUEZ
B.S. University of Central Florida, 2014
M.S. University of Central Florida, 2016

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2018

Major Professor: Sumit Kumar Jha

ABSTRACT

The continued scaling of the CMOS device has been largely responsible for the increase in computational power and consequent technological progress over the last few decades. However, the end of Dennard scaling has interrupted this era of sustained exponential growth in computing performance. Indeed, we are quickly reaching an impasse in the form of limitations in the lithographic processes used to fabricate CMOS processes and, even more dire, we are beginning to face fundamental physical phenomena, such as quantum tunneling, that are pervasive at the nanometer scale. Such phenomena manifests itself in prohibitively high leakage currents and process variations, leading to inaccurate computations. As a result, there has been a surge of interest in computing architectures that can replace the traditional CMOS transistor-based methods. This thesis is a thorough investigation of how computations can be performed on one such architecture, called a crossbar. The methods proposed in this document apply to any crossbar consisting of two-terminal connective devices. First, we demonstrate how paths of electric current between two wires can be used as design primitives in a crossbar. We then leverage principles from the field of formal methods, in particular the area of bounded model checking, to automate the synthesis of crossbar designs for computing arithmetic operations. We demonstrate that our approach yields circuits that are state-of-the-art in terms of the number of operations required to perform a computation. Finally, we look at the benefits of using a 3D crossbar for computation; that is, a crossbar consisting of multiple layers of interconnects. A novel 3D crossbar computing paradigm is proposed for solving the Boolean matrix multiplication and transitive closure problems and we show how this paradigm can be utilized, with small modifications, in the XPoint crossbar memory architecture that was recently announced by Intel.

To my parents, without whose sacrifice this would not have been possible.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Sumit Kumar Jha, for his continued support and encouragement. He was instrumental in providing me with the intellectual freedom and guidance necessary to succeed in this discipline. I would also like to extend my sincerest gratitude to my friend and mentor Dr. K. Subramani for enhancing my knowledge of combinatorial optimization and complexity theory, as well as for providing a respite from the toils of research.

My gratitudes are also with many researchers from the Air Force Research Laboratory. From the autonomous command and control systems branch (RISC), I would like to thank Dr. Robert Wright for hosting and involving me in his cutting-edge research on deep reinforcement learning and Dr. Nathaniel Gemelli and Dr. Jeffrey Hudack for advice and help in the interviewing process and the inner workings of the laboratory. From the trusted systems branch (RITA), I am grateful to Steven Drager for hosting me multiple times and for being constructively critical of my research agenda and Dr. Matthew Anderson for his help and generosity.

Credit is also due to Dr. Nathaniel Cady and Zahiruddin Alamgir for their contributions in making some of the theories in this thesis a physical reality. Finally, I would like to thank Dr. Gary Leavens and Dr. Annie Wu for taking time out of their busy schedules to join my defense committee.

This research would not have been possible without the financial support of the National Science Foundation Graduate Research Fellowship Program (GRFP) and award $\#1438989$ titled "XPS: EXPL: FP: Collaborative Research: Formal methods based algorithmic synthesis of more-than-Moore nano-crossbars for extreme-scale computing", the Air Force Research Laboratory, and my own Alma Mater, the University of Central Florida. The views expressed herein are my own and do not necessarily reflect the views of these great institutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Computing on current high-performance machines is exacerbated by three issues that arise from challenges in the processing and storage of data. First, the era of sustained transistor scaling per Moore's law is now believed to be coming to an end. In the past, such predictions have been made and the obstacles have been overcome by the ingenuity and collaboration of engineers and scientists. However, the lithographic processes required to fabricate ever-smaller transistors result in higher process variations and more unpredictable structures when working at the atomic scale [8]. Furthermore, the alignment of these devices poses an even greater challenge as we traverse deeper into the nanoscale [9].

In response to these difficulties, substantial research has been devoted to novel patterning methods that can extend traditional optical lithography. This research effort has produced methods such as extreme ultra-violet (EUV) lithography, nanoimprint lithography (NIL), maskless lithography (ML2), and bottom-up directed self-assembly (DSA). The last of these methods shows promise as a potential successor to top-down lithography [10]. DSA synthesizes relatively basic primitives with precise shapes that may be used for computation. However, it is not suitable for fabricating complex device topologies that frequently occur in modern circuits [8]. The use of these basic primitive structures necessitates a paradigm shift in our circuit model. Motivated by the success of assembling aligned nanowires [11] and the ease of reconfiguration, attention has shifted to the use of crossbars [12], which consist of perpendicular sets of parallel wires with two-terminal components at each wire junction. These can be easily assembled using DSA [13]. However, the fabrication of crossbars using this approach suffers from high defect rates [8]. These defects occur in the form of stuck-off and stuck-on devices as well as broken wires. The high defect rates make it uneconomical to simply reject defective nanoscale crossbars [14]. Hence, there is a pressing need to develop fault-tolerant computing approaches [15, 16, 17] that can ensure the fruitful use of even

defective nanoscale crossbars. To this end, we propose a new methodology for mapping logical and arithmetic operations onto a defective crossbar that exploits the occurrence of faults.

The second limitation in high-performance computing stems from today's popular DRAM and SRAM memories, which are volatile in that their stored data is lost when the power source is removed from the system. Consequently, these architectures require a constant supply of power to refresh the stored data, resulting in poor energy efficiency. In order to meet the energy demands of the future, we must move away from volatile memory. Fortunately, much work is being done in this area. HP has announced their plans of building *the Machine* with non-volatile memory. An overhaul of the traditional processor-centric computing model to a memory-driven one has also been proposed [18]. Intel and Micron recently unveiled their groundbreaking 3D XPoint™ memory architecture [19] – a 3-dimensional crossbar architecture with two layers of non-volatile memory cells which is orders of magnitude faster than traditional solid-state drives [20]. This technology can already be found on the market in the form of Intel's Optane™ drives [21]. HP and Sandisk have also announced a partnership for investigating the area of memory-driven computing [22] in the hopes of leveraging their memristor and non-volatile resistive RAM technologies.

Though a useful step in the right direction, the adoption of non-volatile technology may not be enough on its own. This brings us to our third problem in high-performance computing: the infamous memory wall [23] that arises in traditional von Neumann architectures where the memory and processing units are segregated. This separation poses serious restrictions on performance. Under a 45 nm process at 0.9 V, it has been reported [24] that 32-bit floating-point multiplication consumes approximately 4 pJ of energy whereas accessing DRAM requires $1.3 - 2.6$ nJ. There is a difference of three orders of magnitude in power consumption between computation and memory access in this case. A study by NVIDIA yielded similar results [25]. Namely, that a 64-bit arithmetic operation consumes 20 pJ of energy as opposed to the 16 nJ consumed during DRAM read/write operations.

The aforementioned problems motivate the need for a non-volatile in-memory computing architecture that uses post-CMOS devices. Such architectures have been studied in the domain of memristor crossbar computing. This powerful alternative to traditional computing uses crossbars to perform both memory and logic operations, thereby eliminating the separation of memory and processing units that is characteristic of traditional architectures and offering a potential solution to the second and third problems mentioned. This topic has garnered significant attention in the past few years due to the reasons mentioned earlier, as well as the breakthrough discovery of memristors [26], which are two-terminal non-volatile memory components that are regarded as potential successors to the transistor. It is worth noting that these memristors have been successfully assembled into crossbars through non-lithographic methods [8, 13]. Thus, in-memory memristor-based computing helps to address the three problems mentioned. It is worth noting that the results presented in this document apply to other memory architectures as well. In particular, any memory whose components store values and act as open or closed switches depending on these values. One such example is phase-change memory (PCM).

The remainder of this thesis is organized as follows.

Chapter 2: A detailed exposition of the three main facets of crossbar architectures is presented. These are the interconnects, their arrangement into crossbar designs, and the sneak paths constraint that arises therein.

Chapter 3: Related work is presented. Namely, we present popular methods for computing Boolean functions using crossbar memories. These include procedures based on logic synthesis, formula-dependent placement of interconnects, and crossbar networks, among others.

Chapter 4: The theory of paths-based logic is presented. This theory provides a framework for arguing about the number of trajectories, or paths, that electric current can flow through between two wires in a crossbar. The notion of irreducible paths is introduced and is used to prove that only

a small subset of paths need to be considered in order to perform computations. The universality of paths-based logic is also established by proposing a general design methodology for any given Boolean formula.

Chapter 5: We leverage the results in Chapter 4 and advances in model checking to propose a framework that generates compact crossbar designs to compute a given Boolean formula. This framework is then extended to handle common crossbar faults such as stuck-on and stuck-off defects as well as broken wires.

Chapter 6: The limitations of using traditional crossbars and paths-based logic to compute functions are proven rigorously. We then demonstrate how such limitations can be overcome by using heterogeneous crossbars. These crossbars augment the traditional model by allowing interconnects to route the flow of current in some particular direction as opposed to restricting ourselves to interconnects that allow bidirectional flow of current across them. A fault-tolerant design automation approach using these heterogeneous crossbars is proposed in a similar fashion to the one in Chapter 5.

Chapter 7: An alternative design automation approach using bounded model checking is presented. As opposed to searching through the space of possible memory configurations, we now search the space of voltage inputs that can be applied on the wires of the memory. Namely, we synthesize the shortest sequence of voltage pulses that will store the result of some desired formula. By allowing the synthesis procedure to perform destructive operations wherein input data can be overwritten, we achieve designs that are state-of-the-art in terms of space.

Chapter 8: We focus our attention on crossbars consisting of 1-diode 1-resistor (1D1R) interconnects and how they can be used to compute the product of Boolean matrices, a fundamental problem in computer science. This problem is of interest for various reasons. It is an integral part of group testing [27] and its applications in genetics, data forensics, fault diagnosis, and on-chip

sensing. Boolean matrix multiplication has also been studied in the context of matrix decomposition [28], cryptography [29], and CFG parsing [30, 31].

Chapter 9: We present a new in-memory computing architecture that can compute the transitive closure of graphs within non-volatile 3D resistive memory (3D RRAM) with external feedback loops. We demonstrate that our approach has a runtime complexity of $\mathcal{O}(n^2)$ operations using $\mathcal{O}(n^2)$ devices. In practice, we argue that our method is both fast and energy-efficient (see Table 9.4 and Section VI).

Chapter 10: We briefly outline the contributions made, suggest potential avenues of research in the area, and allude to future work.

CHAPTER 2: BACKGROUND

Memristors

The idea of a memristor was first theorized by Leon Chua in 1971 based on a fundamental symmetry argument. Chua argued that there were existing relationships between all of the different electromagnetic elements, except for magnetic flux and charge. Hence, he postulated that a relationship between charge and magnetic flux must exist; he defined the component that establishes this relationship as a *memristor*. Memristance, as defined by a nonlinear relationship between voltage and current, has been serendipitously observed in circuits for decades. It was normally considered an anomaly until 2008, when the quantum science research group at Hewlett Packard laboratories developed the first working memristor and theoretical models explaining its behavior.

Since their discovery by HP Labs in 2008, memristors have been a topic of active research due to several desirable properties, including (i) their small size ($< 10\ nm$), (ii) the ability of a memristor to maintain its state for years after bias has been removed, and (iii) their low power footprint compared to traditional transistors.

The memristor, as detailed in [32], is a two-terminal passive component acting as two variable-state resistors in series – one acting as a semiconductor doped with ion vacancies and the other acting as an insulator. The doped region drifts as the result of electric fields generated by the voltage bias. The properties of the memristor are dictated by its material composition. While the TiO_2 film has been well-studied in memristive and resistance-switching devices, HfO_2 [33], a-Si [34], TaO_x [35], a-LSMO [36], and a-SrTiO3[37] have also been studied, among others. These differing electrochemical compositions give rise to varying degrees of nonlinearity in the voltage-current relationship of the memristor. They also determine the endurance and power consumption of the

devices and can even cause interesting effects, such as producing unipolar rectifying memristors that act similarly to diodes [38].

The state of the memristor is uniquely identified by the width w of the doped region of the memristor, as can be seen in Figure 2.1. Here, D is the entire width of the memristor. A positive current causes the doped region to grow, thus increasing w and reducing the overall resistance (memristance) of the memristor. Conversely, a negative current will decrease w and make the resistance of the memristor higher. We denote the low resistance of the doped region by R_{on} and the high resistance of the undoped region by R_{off}.

It is worth noting that a voltage threshold which must be exceeded in order to change the state of a memristor, and therefore its resistance, is apparent when there exists a strong nonlinearity in the I-V relationship of the memristor [6]. This allows us to use voltages smaller than said threshold in order to read values from memristors or memristor crossbars without destroying data by erroneously changing the state of the memristors involved.

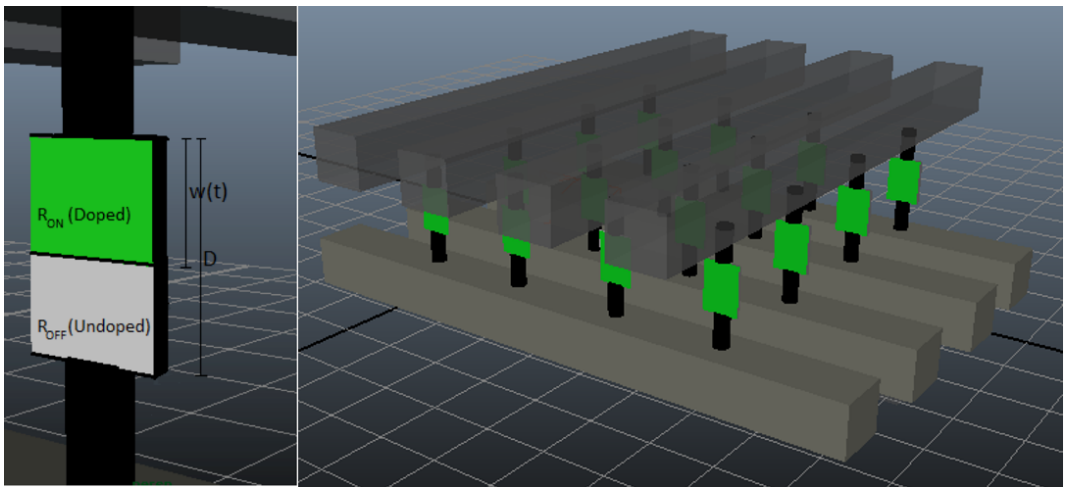


Figure 2.1: A memristor can be modeled as a pair of resistances in series – a low-resistance resistor of width w and a high-resistance resistor of width $D - w$ (left). 4×4 Memristor crossbar (right).

The focus of this thesis is on memristors acting as switches. To this end, we only deal with memristors in one of the extreme states such that their resistance is R_{on} or R_{off}. A memristor M in its lowest (highest) resistance state acts similarly to a closed (open) switch and is said to the in the LRS (HRS) state. Here, LRS and HRS stand for low-resistance and high-resistance state. The state m of memristor M is given by $m = 1$ ($m = 0$) to denote the LRS (HRS) state.

Crossbar Computing

A crossbar consists of two sets of mutually perpendicular wires. Each wire from one set is connected to every wire from the other set through an electronic component. This interconnection can be a two-terminal device such as a diode or memristor, or a three-terminal device such as the CMOS transistor. In the remainder of this section, we will briefly discuss a few popular crossbar computing architectures.

The NanoFabric [39] consists of an array of interconnected and reprogrammable nano-crossbars, called nanoBlocks, with corresponding switch blocks. The NanoFabric is similar to an FPGA *sans* the look-up tables. A number of NanoFabric designs have been proposed that include AND, OR, XOR, and a half-adder. The NanoPLA [40] uses reconfigurable diodes or rectifying diode-like elements in tandem with NOR-NOR logic in order to perform logical operations. The Nanoscale Application-Specific Integrated Circuit (NASIC) architecture [41] is an ASIC that utilizes field-effect transistors and crossbars. An extension of this framework that interfaces with a 3D CMOS stack has also been explored [42].

Crossbar-computing frameworks using traditional CMOS-like logics have been proposed as well. In one such methodology [43], crossbars of p- and n-type FETs and programmable switches are used to evaluate logical conjunctions, disjunctions, and inversions. A hybrid approach, referred to

as CMOL [44], utilizes CMOS and molecular computing in the form of crossbars. The crossbars are fabricated on top of the CMOS die and the two architectures interface via a set of pins that connects the CMOS die to the top and bottom nanowires. Bottom-up self-assembly is perfectly suited for this by allowing the crossbars to grow regardless of substrate used, thus not requiring the high temperatures that restrict the fabrication of CMOS circuitry on top of another CMOS stack [9]. The crossbars could function as memory or as reconfigurable computing fabrics. A variation of CMOL [45] uses a field programmable nanowire interconnect (FPNI) to facilitate communication between the crossbar and the CMOS stack. Some additional constraints are enforced, such as confining logic to the CMOS stack and restricting routing to the crossbar. The resulting architecture trades off speed and fault-tolerance for ease of fabrication when compared to traditional CMOS. A 3D extension of CMOL was introduced in [46]. In this architecture, two CMOS stacks are used and the crossbar is sandwiched between the two stacks. This allows each stack to communicate with only one set of wires, thereby mitigating the pin interfacing difficulties of traditional CMOL.

Promising emerging technologies, such as the memristor [26] and spin-transfer-torque (STT) devices, have also been introduced to the crossbar-computing domain both as memory and as computation nodes. A 3-D crossbar architecture [47] uses memristors together with an interfacing methodology similar to field programmable nanowire interconnect (FPNI). The use of Null Convention Logic (NCL) [48] has also been explored to develop an asynchronous lookup table using a memristor crossbar. Sneak paths [49] pose a problem in these architectures by causing HRS nodes to be read as LRS nodes due to the flow of current through sequences of LRS nodes that run in parallel to the current being used to measure the HRS node. This problem is analogous to the crosstalk problem in RRAM memories. Solutions have been proposed via the use of 1M1D memristor-diode structures or diode-like rectifying memristors [50] [2]. In this thesis, we explore the application of formal methods to the problem of designing crossbars that can exploit these sneak paths as a means for evaluating Boolean formula [51] rather than suppress them.

One omission often made in the literature lies in addressing the robustness of crossbar computing designs in the presence of stuck-off and stuck-on nodes as well as broken wires, which are common defects resulting from self-assembly [8]. Given a faulty $n \times n$ crossbar, one solution [52] determines the $k \times k$ ($k < n$) maximum defect-free subset of the crossbar which can then be used in the application-mapping stage of the fabrication process. However, the computational power of the resulting crossbar can be diminished due to loss of significantly many computation nodes. Another algorithm for finding defect-free subcrossbars from a defective crossbar is proposed in [53]. Other methods include simply avoiding the defective devices altogether by providing redundancy in the form of spare parts [54].

In light of the high defect rates associated with self-assembled crossbars, we propose a new approach to fault-tolerant crossbar computing that exploits the stuck-at interconnects and broken wires to design nanoscale crossbars for computing Boolean formula. As our method does not try to identify a sub-crossbar of defect-free interconnects and instead leverages defects during the design phase, it is capable of performing useful computations even on highly defective crossbars. Our approach is independent of the underlying architecture and only requires the use of a crossbar consisting of digital switches. As such, this framework can be easily implemented using memristor crossbars, CMOL, and phase-change memories (PCMs).

Sneak Paths

One problem that arises in crossbar computing is that of sneak paths. That is, the paths of current that flow through the crossbar which may cause wrong values to be read from memristors. These sneak paths are formally defined in [55] for an $|R|$-by-$|C|$ binary matrix representation of the memristor states in the crossbar. A path of length $2k + 1$ affecting the cell at position (i, j) is said to be a sneak path if $m_{ij} = 0$ (i.e. the memristor connecting wires i and j is in the HRS state) and

there exist $2k$ positive integers $1 \leq r'_1, \ldots, r'_k \leq |R|$ and $1 \leq c'_1, \ldots, c'_k \leq |C|$ for some $k \geq 1$ such that the following $2k + 1$ memristor states satisfy

$$m_{ic'_1} = m_{r'_1c'_1} = m_{r'_1c'_2} = \cdots = m_{r'_{k-1}c'_k} = m_{r'_kc'_k} = m_{r'_kj} = 1 \tag{2.1}$$

Thus, a path of LRS memristors can cause an HRS memristor's interconnected wires to have a significant flow of current across them. This is normally seen as a problem in the literature, especially in individually-addressable memory designs where a current is applied to read the state of a particular memristor. This same current can create a sneak path as mentioned above and cause a wrong state to be read (i.e. an HRS memristor might be read as if it were an LRS memristor). For example, suppose we wish to read the state of the blue memristor in the crossbar in Fig. 2.2, where green and black nodes represent LRS and HRS memristors, respectively. Naturally, a voltage bias is applied at one terminal of the blue memristor while the other terminal is grounded. It follows that, if the blue memristor is in the HRS state, then the voltage drop across the resistor-to-ground should be small. This basic principle can be violated by the sneak paths constraint, which can cause the blue memristor to be read as if it were in the LRS state regardless of its true state.

Numerous solutions and alleviations to the sneak paths problem have been proposed [56, 57, 58, 59, 60, 61, 62, 63, 64]. We briefly present some of these works.

In [59], a multistage method is proposed to accurately read the state of some memristor in a crossbar via a sequence of read and write operations. A total of three read, three write, and one comparison operation is required. The method in [56] also utilizes multiple readings to accurately determine the state of a memristor in the presence of sneak paths. This work improves upon [59] by introducing multipoint access along with the multiple readings in order to reduce the number of read/write operations from six to three. Unfortunately, both of these methods require additional

complex hardware and, perhaps more obtrusively, multiple operations are required for what would ideally be a single operation.

Traditionally, transistors and diodes have been deployed in computer architectures to prevent the propagation of undesired current. Thus, it is sensible to attempt such a deployment in memristor crossbars. [60] proposes such a framework by positing a hybrid memristor-crossbar architecture that utilizes the 1D1M and 1T1M structures, where each memristor requires a diode or a transistor, respectively. The diodes in the 1D1M structure cause the flow of current to travel in only one direction, thereby eliminating the sneak paths constraint. This does, however, introduce a significant problem to the crossbar's writing process, which requires dual polarities for effective writes and erases. The 1T1M structure also has pervasive side effects in the form of significantly decreased crossbar density and increased power consumption [58].

In order to preserve the outstanding fabrication density intrinsic to memristive devices, it is desirable to mitigate the sneak paths constraint without complex added circuitry. We propose a crossbar computing framework that mitigates the sneak paths problem without added circuitry or time-consuming multistage read and write operations. The designs proposed herein exploit sneak paths as first-class design primitives and employ them to perform the proposed computations.

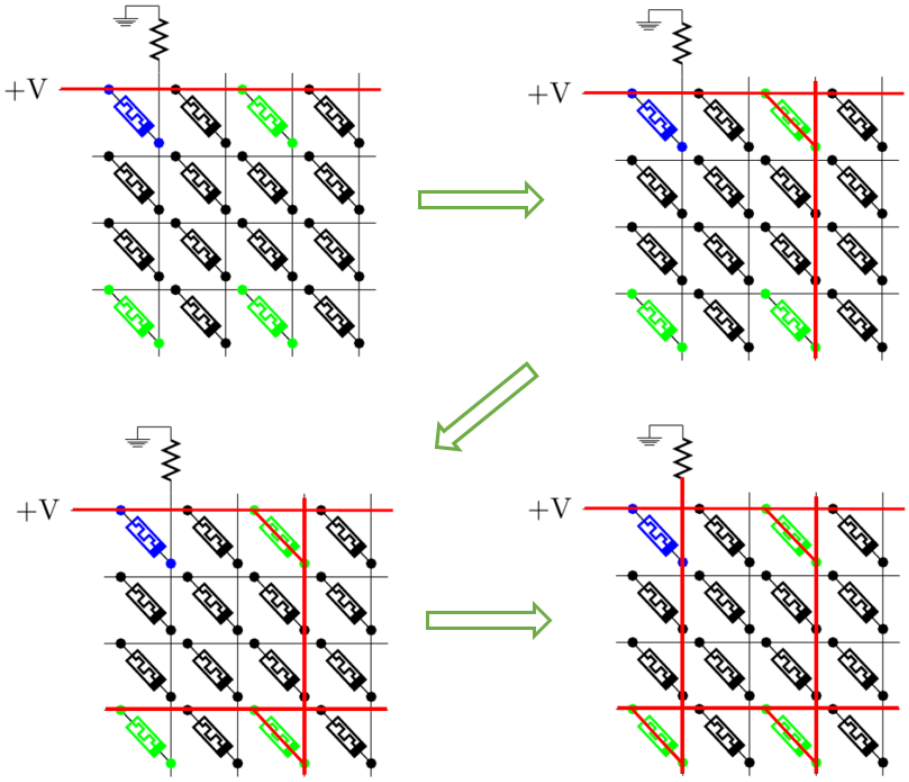


Figure 2.2: Flow behavior of sneak paths, where green nodes denote LRS memristors and black nodes represent HRS memristors. The red bars represent flow of current and we wish to read the value of the blue memristor by applying a voltage at one terminal and grounding the other. It can be seen that a high voltage value will be read at the grounded terminal due to sneak paths regardless of the state of the blue memristor.

CHAPTER 3: LITERATURE REVIEW

Since the discovery of the memristor in 2008, significant effort has been devoted to the development of design methods that allow for memristor-based computation. While there are numerous studies in the area of digital logic computations using individual memristors [65, 66, 67, 68, 2, 69, 3], the literature is scarce when it comes to digital memristor-crossbar computing. This is because developing efficient memristor-crossbar computing frameworks is difficult due to the challenge posed by sneak paths. In this section, we present four memristor-crossbar computing methodologies proposed in the literature and demonstrate how the novel architectures proposed therein overcome the sneak paths constraint and at what cost.

Logic Synthesis Methods

In-memory crossbar computing is largely based on the implication-falsity logic presented in [70]. We briefly present an outline of this procedure. Let P and Q denote two memristors with corresponding states $p, q \in \{0, 1\}$, where 0 and 1 denote high-resistance and low-resistance states (HRS, LRS), respectively. Both devices are connected to the same load resistor R_G and let $V_{\text{COND}}, V_{\text{TH}}, V_{\text{SET}}$ denote voltages such that V_{SET} is sufficient to switch a memristor to the LRS state, V_{TH} is the threshold voltage value that must be exceeded in order to switch said memristor, and $V_{\text{SET}} - V_{\text{COND}} < V_{\text{TH}}$. Apply voltage bias V_{COND} to P and V_{SET} to Q. This will cause Q to switch to or remain in the LRS state if $p \implies q$. See Fig. 3.1 for a visualization of the case when $p = 1$ and $q = 0$. It is easy to see that Q will be switched to or remain in the LRS state under the other 3 evaluations of p and q because the voltage drop across Q will be $V_{\text{SET}} > V_{\text{TH}}$. A voltage V_{CLEAR} can be used to set the memristor to the HRS state corresponding to a binary value of 0, or *false*. This combination of implication and falsity is functionally complete for Boolean operations.

This result catalyzed a host of design methodologies [71, 69, 67, 68] based on minimizing the number of implication operations required to compute a formula of interest.

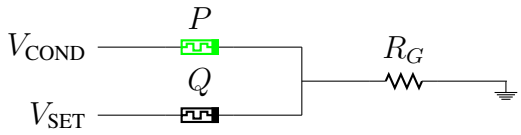


Figure 3.1: Memristor implication logic. The green memristor P is in the LRS state ($p = 1$) and the black memristor Q is in the HRS state ($q = 0$). The voltage drop across Q will be approximately $V_{\text{SET}} - V_{\text{COND}} < V_{\text{TH}}$, so Q will remain in the HRS state, meaning that $q = 0 = (p \implies q)$.

In [72] and [73], each interconnect is a complementary resistive switch consisting of two memristors of opposite polarity. The basic logical operation on these interconnects follows the equation $Z^t = (w \impliedby b) \wedge Z^{t-1} \vee (w \not\Longrightarrow b) \wedge \neg Z^{t-1}$, where Z^t is the state of the device at time t, w (b) is the wordline (bitline) such that $w = 1$ ($b = 1$) in the presence of a high voltage potential and 0 otherwise. This allows greater control over the flows of current throughout the crossbar and it is demonstrated how under this approach an n-bit NAND operation can be computed in a constant number of steps as opposed to the linear amount demonstrated in previous works. The results in [73] apply specifically to the design of adders while the contribution in [72] is a general design methodology using NOR-OR and NAND-AND logic.

In [74], a method for computing is proposed that deviates from the omnipresent implication-falsity logic. Each gate consists of a set of input memristors programmed to the HRS or LRS state and an output memristor that will be programmed to the LRS (HRS) state if the gate operation evaluates to 1 (0). Each input memristor is configured to the LRS (HRS) state if its corresponding variable is 1 (0). A programming voltage is then applied so that the output memristor will reach the LRS (HRS) state if the underlying operation evaluates to 1 (0). For example, to perform an n-bit disjunction (OR), n input memristors are placed in parallel and connected to the output memristor. It follows that if any of the input memristors is in the LRS state, then the output memristor connected to it

will be switched to the LRS state, denoting a value of 1 as intended. Logical conjunction (AND) is performed by placing the input memristors in series so that all of them must be in the LRS state in order to switch the output memristor. Similar procedures are used to compute NAND, NOR, and NOT. It is worth noting, however, that only the NOR operation is amenable to integration within a traditional crossbar array. As such, a computing paradigm using this approach would necessitate additional structural complexities.

The methods referenced in this section are used as benchmarks against which we compare the complexity of some of our proposed designs (See Table 6.1). In particular, we demonstrate that our design is state-of-the-art in terms of execution steps required to compute addition. Our approach is fundamentally different from the methods mentioned in that the components in the crossbar are programmed only once in such a way that the induced trajectories of current can be used to compute.

Formula-Dependent Memristor Placement

In [1], a crossbar construction for evaluating Sum-of-Minterms form Boolean formulas is proposed. The method consists of mapping each clause in the formula to a separate column in the crossbar and mapping all of the literals to the first column. Every column, except the first, contains an LRS memristor at the bottommost row in order to redirect current during the computation step. To perform the actual computation, a read-out voltage is applied on the first column and the formula evaluates to true if one of the columns redirects current to its corresponding LRS memristor, which in turn would use this current to determine and save the state of the result memristor. The design manages sneak paths by removing memristors from locations that are uniquely determined by the formula being evaluated.

As an example, we construct the 3-bit parity function $(b_1 \wedge b_2 \wedge \neg b_3) \vee (b_1 \wedge \neg b_2 \wedge b_3) \vee (\neg b_1 \wedge b_2 \wedge b_3) \vee (b_1 \wedge b_2 \wedge b_3)$ using this method in Figure 3.2.

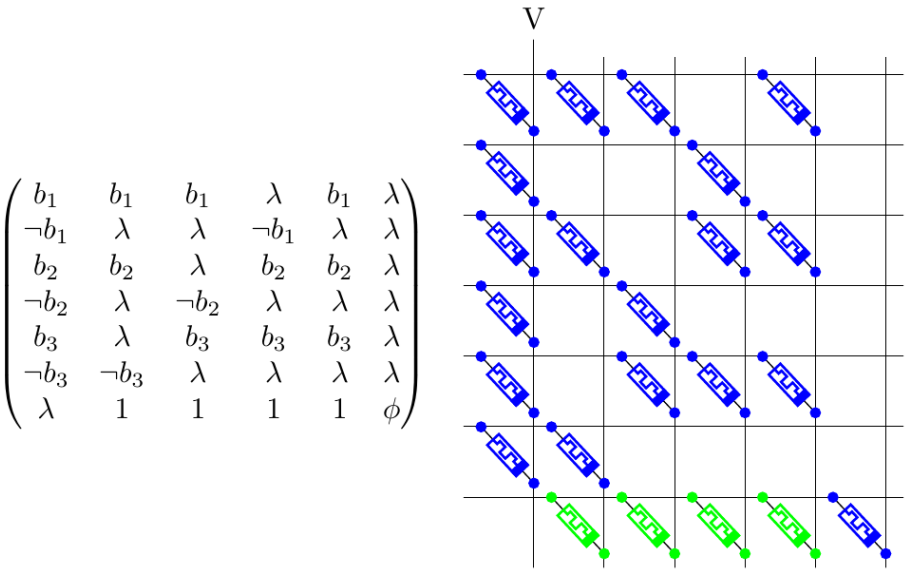


Figure 3.2: Matrix (left) and crossbar (right) representations of a 3-bit parity construction using the design proposed in [1], where green memristors are in the LRS state, λ values specify a location in which there is no memristor, and the states of blue memristors are determined by variables b_i and their negations $\neg b_i$ in the Boolean formula ϕ which we wish to evaluate. The resulting value of ϕ will be stored in the bottom right memristor of the crossbar when the computation is finished.

While this approach benefits from fast computation time, it overcomes the sneak paths issue by adding significant complexity to the design, where only certain junctions may contain memristors and the locations of said junctions vary from formula to formula. This can lead to prohibitively expensive fabrication costs as well as difficulty in reconfiguring such crossbars.

Rectifying-Memristor Crossbars

Another crossbar computing method is introduced in [2]. This method leverages the diode-like capabilities of rectifying memristors [50] to suppress the interference from sneak currents. Unlike

the previous design, there is no added complexity to the crossbar beyond the use of rectifying memristors.

A new operation is introduced based on a procedure similar to the implication logic process described in [66]. This operation is called converse nonimplication, and it provides a method for configuring a crossbar of rectifying, diode-like memristors as well as for performing logical functions. This memristive converse nonimplication logic is used in tandem with the multi-input implication logic introduced in [68] to configure the crossbar and compute the Boolean formula of interest. In the case of a 3-bit parity function ϕ, we can map the formula to a 5-by-4 crossbar as follows:

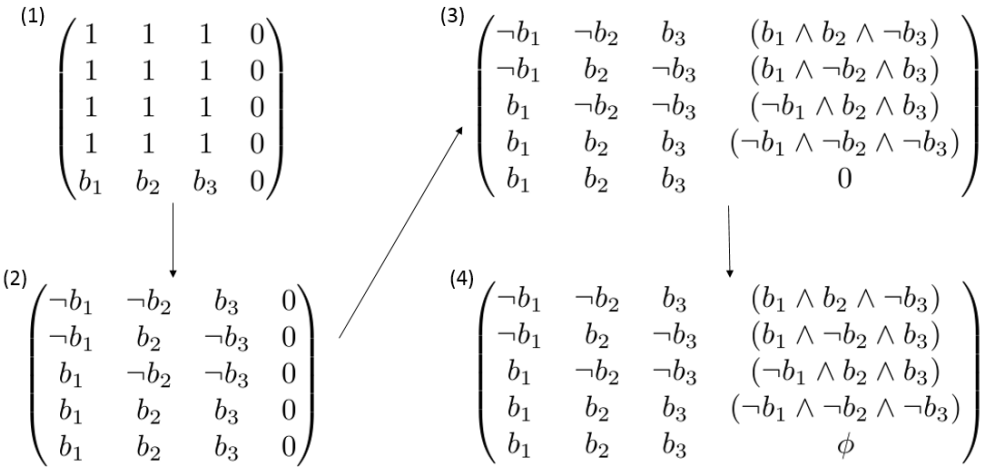


Figure 3.3: States of the 3-bit parity function over inputs $b_1, b_2, b_3 \in \{0, 1\}$ mapped to a crossbar using the method in [2].

In Figure 3.3, we assume that the crossbar is initialized to state (1) in a constant number of steps. State (2) is obtained in $2n$ steps via converse nonimplication, where n is the number of literals in ϕ. State (3) can then be achieved in a single time step by performing a multi-input implication operation on rows 1 through 4 in parallel. This operation is possible thanks to the unidirectional sneak current flow provided by the rectifying memristors. Note that the resulting literal values in column 4 are the negation of the literals in columns 1 through 3; this is because the implication

operation $M_{i4} = ((b_1 \vee b_2 \vee b_3) \implies 0) = (\neg b_1 \wedge \neg b_2 \wedge \neg b_3)$. Similarly, applying a multi-input implication step on column 4 yields state (4), where $\phi = (\neg b_1 \vee \neg b_2 \vee b_3) \wedge (\neg b_1 \vee b_2 \vee \neg b_3) \wedge (b_1 \vee \neg b_2 \vee \neg b_3) \wedge (b_1 \vee b_2 \vee b_3)$ is stored in $m_{5,4}$. This method benefits from the use of a simple crossbar layout and its intuitive use of rectifying memristors to mitigate the sneak paths problem. However, having to reduce any formula to a sometimes complex sequence of implication logic operations can lead to considerable overhead. For example, a 3-bit parity function requires 30 computational steps to be evaluated using elementary implication logic [75].

Networks of Interconnected Crossbars

Finally, we have the framework proposed in [3]. This method can be seen as an extension of [76] using a network of interconnected crossbars in order to improve the space complexity of the resulting designs. Any individual crossbar in the network can be used to evaluate a conjunction or disjunction of Boolean variables. By connecting multiple crossbars together, disjunctions (conjunctions) of conjunctions (disjunctions) can be evaluated. Thus, the universal disjunctive and conjunctive normal forms (DNF, CNF) can be computed. See Figure 3.4 for the DNF and CNF designs and Figure 3.5 for an example.

The methods presented in this section succeed in computing Boolean formula using memristor crossbars. These methods are diverse and have different benefits and drawbacks; however, none of them meet all of the following three fundamental tenets of computing architectures: space efficiency, fast computation time, and cost-effectiveness due to structural simplicity.

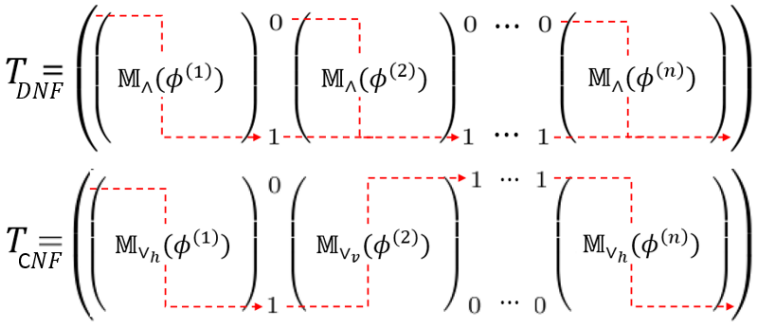


Figure 3.4: Red dashed lines denote the flow of current and $\mathbb{M}_*(\phi^{(i)})$ represents the crossbar that computes the i^{th} clause of the Boolean formula ϕ, which must be in either disjunctive normal form (top) or conjunctive normal form (bottom).

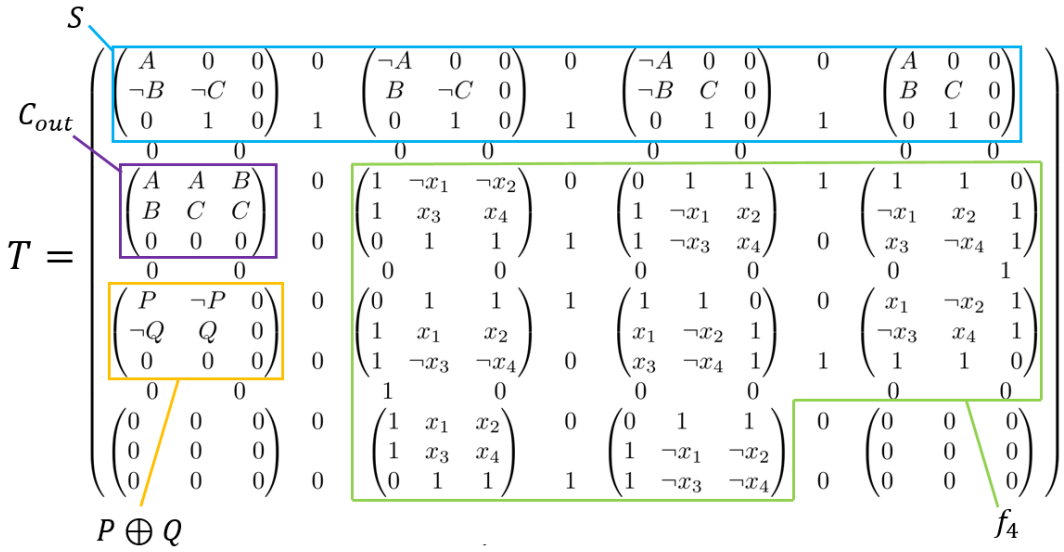


Figure 3.5: Network of crossbars method described in [3]. Here, $S = (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$ and $C_{out} = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$ are the sum and carry-out bits over inputs $A, B, C \in \{0, 1\}$, and f_4 is a 4-bit parity function over inputs $x_1, x_2, x_3, x_4 \in \{0, 1\}$.

CHAPTER 4: PATHS-BASED LOGIC

We begin by formalizing the rather intuitive notion of a crossbar. A pictorial representation of the definition below can be seen in Figure 4.1.

Definition 1. CROSSBAR *An $|R| \times |C|$ crossbar is a 3-tuple $\mathcal{X} = (M, R, C)$ where*

- $M = \begin{pmatrix} M_{11} & M_{12} & \dots & M_{1|C|} \\ \vdots & \vdots & \ddots & \vdots \\ M_{|R|1} & M_{|R|2} & \dots & M_{|R||C|} \end{pmatrix}$ *represents a matrix of interconnects with $|R|$ rows and $|C|$ columns. $m_{ij} \in \{0, 1\}$ denotes the state of device M_{ij} connecting row i with column j. $m_{ij} = 0$ ($m_{ij} = 1$) denotes a device in the HRS (LRS) state.*

- $R = \{R_1, \dots, R_{|R|}\}$ *is the set of row wires, where $r_i \in \{0, 1\}$ provides the same input voltage to every interconnect in row R_i. $r_i = 0$ ($r_i = 1$) denotes the absence (presence) of electric current in wire R_i.*

- $C = \{C_1, \dots, C_{|C|}\}$ *is the set of column wires, where $c_j \in \{0, 1\}$ provides the same input voltage to every interconnect in column C_j. $c_j = 0$ ($c_j = 1$) denotes the absence (presence) of electric current in wire C_j.*

For the remainder of this thesis, we utilize capital letters to signify wires and components and lowercase letters to denote their value. For example, we say that a wire $W \in R \cup C$ has value $w = 0$ if there is negligible current on the wire and $w = 1$ if there is a significant amount.

Irreducible Paths in Crossbars

It is clear that if there is an electric potential difference between two wires R_α and R_β, then there must exist a sequence of interconnects that, when in the LRS state, would generate a current from R_α to R_β. We refer to this sequence of nodes as a path. Any such path is represented by the ordered set $\Pi^{R_\alpha \to R_\beta} = \{M_{\alpha j_1}, M_{i_1 j_1}, M_{i_1 j_2}, \ldots, M_{i_k j_l}, M_{\beta j_l}\}$, where each $M \in \Pi^{R_\alpha \to R_\beta}$ is a constituent component in the path. It is obvious that the first node resides on wire R_α and the last node must reside on R_β. For the sake of simplicity, we will assume that the source and destination of a path are both row wires.

Definition 2 (Path)**.** *A path $\Pi^{R_\alpha \to R_\beta} = \{M_{\alpha j_1}, M_{i_1 j_1}, M_{i_1 j_2}, \ldots, M_{i_k j_l}, M_{\beta j_l}\}$ is an ordered set of nodes containing a source node $M_{\alpha j_1}$ and a destination node $M_{\beta j_l}$.*

Axiom 1 (Flow Axioms)**.** *Let $\mathcal{X} = (M, R, C)$ be an $|R| \times |C|$ crossbar. Then the following always hold:*

- $\forall i \leq |R|, j \leq |C|, (r_i \wedge m_{ij}) \implies c_j$.
- $\forall i \leq |R|, j \leq |C|, (c_j \wedge m_{ij}) \implies r_i$.

The basis of our approach lies in computing Boolean formula in a crossbar by mapping the variables of the formula to individual interconnects in the crossbar such that there will be a path of electric current between two wires if and only if the formula we wish to compute evaluates to 1. In Figure 4.1, bits ϕ_1, ϕ_2, and ϕ_3 are mapped to M_{11}, M_{23}, and M_{34}, respectively, while M_{21}, M_{33}, and M_{44} are in the LRS state (i.e. $m_{21} = m_{33} = m_{44} = 1$). Note that this mapping creates the path $\Pi^{R_1 \to R_4} = \{M_{11}, M_{21}, M_{23}, M_{33}, M_{34}, M_{44}\}$ which directs flow from wire R_1 to wire R_4 iff $\phi = (\phi_1 \wedge \phi_2 \wedge \phi_3) = 1$ given $r_1 = 1$, i.e. an initial flow of current on R_1.

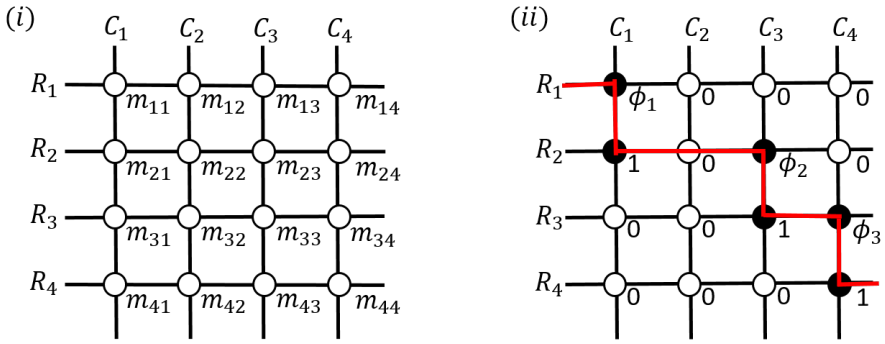


Figure 4.1: (i) 4×4 crossbar $\mathcal{X} = (M, R, C)$. (ii) Mapping of the formula $\phi = (\phi_1 \wedge \phi_2 \wedge \phi_3)$ onto M, where R_1 and R_4 are the source and destination wires, respectively.

Definition 3 (Irreducible Path)**.** *Let* $\tilde{\Pi}^{R_\alpha \to R_\beta}$ *denote the set of all paths from* R_α *to* R_β *in a crossbar.* $\Pi \in \tilde{\Pi}^{R_\alpha \to R_\beta}$ *is an irreducible path if and only if* $\forall \Pi' \in \tilde{\Pi}^{R_\alpha \to R_\beta}, \Pi' \not\subset \Pi$.

According to Definition 3, a path is irreducible if no other path with the same source and destination nodes is contained within it. See Fig. 4.2 for an example. Path $\Pi^{R_1 \to R_8} = \{M_{11}, M_{41}, M_{43}, M_{63}, M_{68}, M_{28}, M_{26}, M_{46}, M_{45}, M_{85}\}$ is not an irreducible path because path $\Pi' = \{M_{11}, M_{41}, M_{45}, M_{85}\}$ is contained within it. That is, $\Pi' \subset \Pi^{R_1 \to R_8}$.

Graphs can be used to represent paths in crossbars. The vertices in this graph represent the row and column wires while the edges correspond to the interconnects connecting these wires. The unweighted connected digraph $G = (V, E)$ corresponding to $\Pi^{R_1 \to R_8}$ is shown in Fig. 4.2b. We define $\Pi_G^{R_1 \to R_8} = \{(R_1, C_1), (C_1, R_4), (R_4, C_3), (C_3, R_6), (R_6, C_8), (C_8, R_2), (R_2, C_6), (C_6, R_4), (R_4, C_5), (C_5, R_8)\}$ as a path in the graph. Note that $\Pi_G^{R_1 \to R_8}$ and $\Pi^{R_1 \to R_8}$ are equivalent representations of the same path. We distinguish between the interconnect and edge representations $\Pi = \{M_{ij}\}_{i,j}$ and $\Pi_G = \{(R_i, C_j), (C_j, R_k)\}_{i,j,k}$ of a path Π by adding the subscript G. Note that the graph formed by $\Pi_G^{R_1 \to R_8}$ has a cycle consisting of the edges in $\Pi_G^{R_1 \to R_8} \setminus \Pi'_G$. Furthermore, the irreducible path $\Pi'_G = \{(R_1, C_1), (C_1, R_4), (R_4, C_5), (C_5, R_8)\}$ contains no cycles.

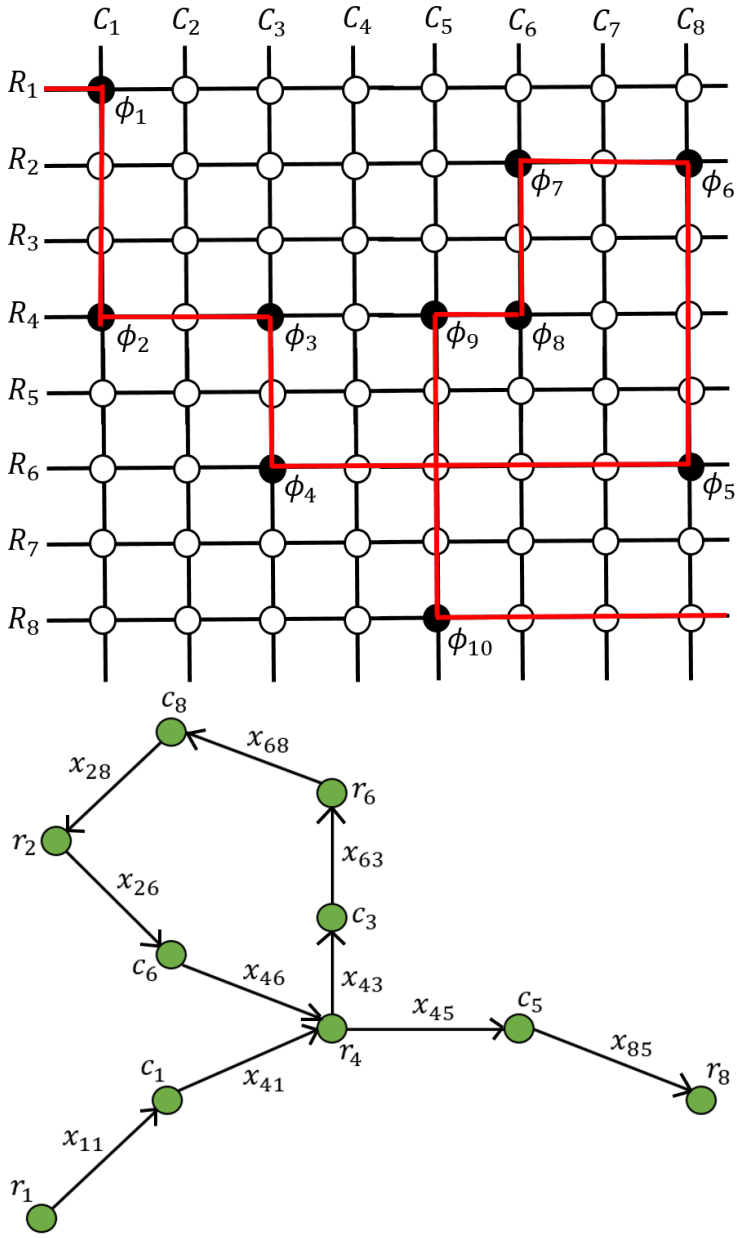


Figure 4.2: (*top*) 8×8 crossbar mapping the formula $\phi = (\phi_1 \wedge \phi_2 \wedge \phi_9 \wedge \phi_{10})$. The red bars represent the path $\Pi^{R_1 \to R_8} = \{M_{11}, M_{41}, M_{43}, M_{63}, M_{68}, M_{28}, M_{26}, M_{46}, M_{45}, M_{85}\}$ corresponding to $\bigwedge_{i=1}^{10} \phi_i$. (*bottom*) Equivalent representation of $\Pi^{R_1 \to R_8}$ as a graph given by $\Pi_G^{R_1 \to R_8} = \{(R_1, C_1), (C_1, R_4), (R_4, C_3), (C_3, R_6), (R_6, C_8), (C_8, R_2), (R_2, C_6), (C_6, R_4), (R_4, C_5), (C_5, R_8)\}$, where $(R_i, C_j) \in \Pi_G^{R_1 \to R_8}$ denotes a directed edge from R_i to C_j.

Theorem 1. *A path $\Pi^{R_\alpha \to R_\beta}$ in an $|R| \times |C|$ crossbar $\mathcal{X} = \{M, R, C\}$ is reducible if and only if $\Pi_G^{R_\alpha \to R_\beta}$ contains a cycle.*

Proof. ($\Longrightarrow$) We argue the contrapositive. Let $\Pi_G^{R_\alpha \to R_\beta} = \{(R_\alpha, C_{j_1}), (C_{j_1}, R_{i_1}), \ldots, (R_{i_k}, C_{j_l}), (C_{j_l}, R_\beta)\}$. Since there are no cycles, $\Pi_G^{R_\alpha \to R_\beta}$ is a minimally connected graph i.e. the out- and in-degrees of all nodes are at most 1. Thus, by definition of minimally connected graphs, there is only one path between any two nodes. It follows that there is only one path from R_α to R_β in the graph formed by $\Pi_G^{R_\alpha \to R_\beta}$. Therefore, the path is irreducible.

($\Longleftarrow$) Suppose, without loss of generality, that $\Pi = \{M_{i_1 j_1}, M_{i_2 j_1}, M_{i_2 j_2}, \ldots, M_{i\alpha_1}, M_{i\alpha_2}, \ldots, M_{i\alpha_3}, M_{i\alpha_4}, \ldots, M_{i\alpha_{2k-1}}, M_{i\alpha_{2k}}, \ldots\}$ is an irreducible path containing a cycle(s). We can construct $\Pi' = \Pi \setminus \{M_{i\alpha_2}, \ldots, M_{i\alpha_{2k-1}}\} \subset \Pi$. Thus, since $\Pi' \subset \Pi$, Π violates the irreducible paths property, a contradiction. Thus, paths that contain cycles are reducible. □

A reducible path will carry a flow of current from its source to its destination wire whenever the irreducible path contained in it can carry a flow of current from the source to the destination. Thus, the presence of reducible paths in crossbar designs does not permit the crossbar to compute new Boolean formula.

Corollary 1. *A crossbar design using only irreducible paths and a crossbar design using all paths can evaluate the same set of Boolean formula.*

Proof. It follows from *Theorem 1* that reducible paths have cycles and irreducible paths do not. Suppose we are given an n-ary Boolean formula $\phi(\phi_1, \ldots, \phi_n)$ and paths $\Pi^{R_\alpha \to R_\beta}$ and $\Pi'^{R_\alpha \to R_\beta}$, such that $\Pi'^{R_\alpha \to R_\beta} \subset \Pi^{R_\alpha \to R_\beta}$. In the context of our framework, we say that ϕ evaluates to true if and only if there is a path $\Pi^{R_\alpha \to R_\beta}$ from R_α to R_β or, equivalently, if $\Pi_G^{R_\alpha \to R_\beta}$ contains a connected walk from R_α to R_β. Since the inclusion or exclusion of edges in a cycle does not affect

the connectedness of the walk in the graph, it follows that $\Pi^{R_\alpha \to R_\beta}$ is a connected walk if $\Pi'^{R_\alpha \to R_\beta}$ is a connected walk from R_α to R_β. □

Since reducible paths do not add to the expressive power of crossbars, our synthesis technique relies on searching through the space of irreducible paths only. This allows us to prune the search space of our approach substantially by ignoring the redundant reducible paths.

Theorem 2. *A path $\Pi^{R_\alpha \to R_\beta}$ forms a cycle if and only if there are more than two nodes in the path that reside on the same wire in the crossbar.*

Proof. ($\Longrightarrow$) Assume that $\Pi_G^{R_\alpha \to R_\beta} = \{(R_i, C_j), (C_j, R_k)\}_{i,j,k}$ contains a cycle. It follows from *Definition 1* that (R_α, C_{i_1}) is the only outgoing edge from R_α and (C_{i_k}, R_β) is the only incoming edge to R_β. Since $\Pi_G^{R_\alpha \to R_\beta}$ forms a cyclic connected digraph, and R_α and R_β cannot be nodes in a cycle, there must exist some node R_i that belongs to a cycle as well as to a simple walk from R_α to R_β. It follows that there exist edges (C_{j_1}, R_i) and (R_i, C_{j_2}) connecting R_i to the simple walk from R_α to R_β and edges (C_{j_3}, R_i) and (R_i, C_{j_4}) connecting R_i to the cycle. See node R_4 in Figure 4.2b for an example. These edges correspond to interconnects M_{ij_1}, M_{ij_2}, M_{ij_3}, and M_{ij_4} in $\Pi^{R_\alpha \to R_\beta}$, leading to more than two nodes on the same wire.

($\Longleftarrow$) Without loss of generality, assume that $\Pi^{R_\alpha \to R_\beta} = \{M_{\alpha j_1}, M_{i_1 j_1}, M_{i_2 j_1}, M_{i_2 j_2}, \ldots, M_{i' p_1}, M_{i' p_2}, \ldots, M_{i' p_3}, M_{i' p_4}, \ldots, M_{i' p_{2k-1}}, M_{i' p_{2k}}, \ldots, M_{\beta c_l}\}$ has more than two nodes on $R_{i'}$. Note that for all $R_i \neq R_{i'}$, the out- and in-degrees are 1, while the out- and in-degrees of $R_{i'}$ are greater than or equal to 2. Since R_α is the only node with no incoming edges, R_β is the only node with no outgoing edges, and all paths form connected digraphs, it follows that node $R_{i'}$ must be in a cycle. □

Corollary 2. *A path Π is irreducible if and only if there are no more than two nodes on any wire.*

We now have the foundations necessary to enumerate irreducible paths in a crossbar. The length and number of such paths is discussed below.

Lemma 1. *The maximum length* $|\Pi^{R_\alpha \to R_\beta}|_{max}$ *of an irreducible path* $\Pi^{R_\alpha \to R_\beta} = \{M_{\alpha j_1}, M_{i_1 j_1}, M_{i_1 j_2}, \ldots, M_{i_k j_l}, M_{\beta j_l}\}$ *in an* $|R| \times |C|$ *crossbar* $\mathcal{X} = (M, R, C)$ *is* $2(min\{|R| - 1, |C|\})$.

Proof. It follows from *Definition 1* that $M_{\alpha j_1}$ and $M_{\beta j_l}$ are in the path and are the only nodes on R_α and R_β, respectively. Let $R' = R \setminus \{R_\alpha, R_\beta\}$ denote the set of remaining rows, where nodes of $\Pi^{R_\alpha \to R_\beta}$ reside. Note that $|R'| = |R| - 2$. It follows from *Corollary 1* that there can only be up to two nodes per wire. Two cases arise:

- *Case 1* ($|C| \geq |R| - 1$): Each row in R' can have two nodes. Thus, $|\Pi^{R_\alpha \to R_\beta}|_{\text{max}} = 2|R'| + 2 = 2(|R| - 2) + 2 = 2(|R| - 1)$.
- *Case 2* ($|C| < |R| - 1$): Assigning two nodes to each column yields $|\Pi^{R_\alpha \to R_\beta}|_{\text{max}} = 2|C|$ and it follows from *case 1* that assigning two nodes to each row yields $|\Pi^{R_\alpha \to R_\beta}|_{\text{max}} = 2(|R| - 1)$. However, since $|C| < |R| - 1$, it is not possible to have an irreducible path of length $2(|R| - 1)$ as this would require some column to contain more than two nodes, thereby violating the irreducible paths property. Thus, $|\Pi^{R_\alpha \to R_\beta}|_{\text{max}} = 2|C|$. □

Theorem 3. *There are* $\frac{|C|!(|R|-2)!}{(|C|-k/2)!(|R|-1-k/2)!}$ *irreducible paths of length* k *from* R_α *to* R_β *in an* $|R| \times |C|$ *crossbar.*

Proof. Let $f^{(k)}$ denote the number of k-length paths and $\tilde{\Pi}^{(k)}$ denote the set of said paths. For $k = 2$, $\tilde{\Pi}^{(k)} = \{(M_{\alpha j}, M_{\beta j})\}_{j=1}^{|C|}$ for source R_α and destination R_β. Thus, $f^{(2)} = |C|$. For $k = 4, 6, \ldots, |\Pi^{R_\alpha \to R_\beta}|_{\text{max}}$, there are $k - 1$ possibilities for choosing row and column coordinates of the nodes in the path. Since our initial and final choices of row coordinate are fixed to the source and destination wires R_α and R_β, there will be one more choice of column coordinate than

row coordinate. There are $\left\{ {|R|-2 \atop k/2-1} \right\}$ ways to choose row coordinates and $\left\{ {|C| \atop k/2} \right\}$ ways to choose column coordinates, where $\left\{ {|C| \atop k} \right\}$ denotes the permutation of k elements from a $|C|$-element set. Thus, we have $f^{(k)} = \left\{ {|C| \atop k/2} \right\} \left\{ {|R|-2 \atop k/2-1} \right\} = \frac{|C|!(|R|-2)!}{(|C|-k/2)!(|R|-1-k/2)!}$.

Base case: Paths of length 2.

$$f^{(2)} = \frac{|C|!(|R|-2)!}{(|C|-1)!(|R|-2)!} = |C|$$

Inductive hypothesis: Paths of length k.

$$f^{(k)} = \frac{|C|!(|R|-2)!}{(|C|-k/2)!(|R|-1-k/2)!}$$

Inductive step: Note that, given the value of $f^{(k)}$ for any k, there will be $(|R|-2)-(k/2-1) = |R|-1-k/2$ possible row choices and $|C|-k/2$ possible column choices for a $(k+2)$-length path. Therefore,

$$\begin{aligned} f^{(k+2)} &= f^{(k)}(|C|-k/2)(|R|-1-k/2) \\ &= \frac{|C|!(|R|-2)!}{(|C|-k/2)!(|R|-1-k/2)!}(|C|-k/2)(|R|-1-k/2) \\ &= \frac{|C|!(|R|-2)!}{(|C|-k/2-1)!(|R|-k/2-2)!} \\ &= \frac{|C|!(|R|-2)!}{(|C|-\frac{k+2}{2})!(|R|-1-\frac{k+2}{2})!} \end{aligned}$$

Hence, proved by mathematical induction. □

Corollary 3. *The number of irreducible paths from R_α to R_β in an $|R|\times|C|$ crossbar is $T(|R|,|C|) = \sum_{k=1}^{min\{|R|-1,|C|\}} \left\{ {|C| \atop k} \right\} \left\{ {|R|-2 \atop k-1} \right\}$, where $\left\{ {|C| \atop k} \right\}$ denotes the permutation of k elements from an $|C|$-element set.*

Proof. Recall that $|\Pi^{R_\alpha \to R_\beta}|_{\max}$ denotes the maximum length of a path from R_α to R_β. It follows directly from *Theorem 3* and *Lemma 1* that the total number of irreducible paths is

$$\begin{aligned}
\sum_{k=2,\ k \text{ even}}^{|\Pi^{R_\alpha \to R_\beta}|_{\max}} f^{(k)} &= \sum_{k=2,\ k \text{ even}}^{|\Pi^{R_\alpha \to R_\beta}|_{\max}} \frac{|C|!(|R|-2)!}{(|C|-k/2)!(|R|-1-k/2)!} \\
&= \sum_{k=1}^{|\Pi^{R_\alpha \to R_\beta}|_{\max}/2} \frac{|C|!(|R|-2)!}{(|C|-k)!(|R|-1-k)!}. \\
&= \sum_{k=1}^{\min\{|R|-1,|C|\}} \left\{ \begin{matrix} |C| \\ k \end{matrix} \right\} \left\{ \begin{matrix} |R|-2 \\ k-1 \end{matrix} \right\} = T(|R|,|C|)
\end{aligned}$$

□

The number of paths established in the preceding corollary grows rapidly in tandem with the size of the crossbar. While this enormous number of potential paths of current through the crossbar presents a great opportunity for the design of compact computing crossbars, it also poses a considerable obstacle to the invention of spatially-efficient designs via human reasoning. Hence, we suggest the use of automated synthesis techniques for designing nanoscale crossbars for computing Boolean formula.

Universality of Paths-Based Logic

An n-ary Boolean function maps an n-tuple of Boolean values to a Boolean value. It can be defined by a truth table of 2^n rows, one for each possible value an n-tuple may take. Alternatively and more concisely, it can be defined in terms of a few Boolean operators or connectives. A set of Boolean connectives is complete if every Boolean function can be defined by an expression that uses only the connectives in that set. The set consisting of the $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction) connectives is complete. So any Boolean function can be defined by a well-formed

formula (wff) constructed as follows: (i) a Boolean (propositional) variable p is a wff; (ii) if ϕ is a wff, $\neg\phi$ is a wff; (iii) if ϕ_1 and ϕ_2 are wffs, $\phi_1 \wedge \phi_2$ is a wff, and (iv) if ϕ_1 and ϕ_2 are wffs, $\phi_1 \vee \phi_2$ a wff.

Our method to prove the universality of paths-based logic requires the formulas to be in negation normal form (NFF), which is defined by a slight change to these rules. The negation connective may not be applied to an arbitrary wff, but only to a propositional variable. A formula is in negation normal form if it is constructed as follows: (i) a literal, that is, a propositional variable p or $\neg p$, is in NNF, (ii) if ϕ_1 and ϕ_2 are in NNF, $\phi_1 \wedge \phi_2$ is in NNF, and (iii) if ϕ_1 and ϕ_2 are in NNF, $\phi_1 \vee \phi_2$ is in NNF.

Any wff constructed using only connectives $\neg$, $\wedge$ and $\vee$ can be transformed into an equivalent formula in negation normal form by repeatedly applying the De Morgan Laws: $\neg(p \wedge q) \equiv \neg p \vee \neg q$, and $\neg(p \vee q) \equiv \neg p \wedge \neg q$, and simplifying $\neg\neg p$ to p. Thus, the method presented below can be used to evaluate any Boolean function.

Let ϕ be a Boolean formula in negation normal form. We define the state of an $|R| \times |C|$ memristor crossbar M_ϕ as follows, where R^i, C^i denote row/column wires of M_{ϕ_i} when there are two formulas ϕ_1, ϕ_2.

- Literal: For $\phi = a$, a literal, we have

$$M_a = \begin{pmatrix} a \\ 1 \end{pmatrix}$$

- Conjunction: For $\phi = \phi_1 \wedge \phi_2$, the array has $|R^1| + |R^2| - 1$ rows, where M_{ϕ_1} and M_{ϕ_2} share row $R_{|R^1|}$, and $|C^1| + |C^2|$ columns. It is defined by:

$$\mathbb{M}_{\phi_1 \wedge \phi_2} = \begin{pmatrix} \begin{pmatrix} & & \\ & M_{\phi_1} & \\ & & \end{pmatrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{pmatrix} & & \\ & M_{\phi_2} & \\ & & \end{pmatrix} \end{pmatrix}$$

- Disjunction: For $\phi = \phi_1 \vee \phi_2$, the array has $|R^1| + |R^2|$ rows and $|C^1| + |C^2| + 2$ columns, and is defined by:

$$\mathbb{M}_{\phi_1 \vee \phi_2} = \begin{pmatrix} 1 & \begin{pmatrix} & & \\ & M_{\phi_1} & \\ & & \end{pmatrix} & \begin{matrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{matrix} \\ 0 & & \begin{matrix} 0 & \cdots & 0 & 1 \end{matrix} \\ \begin{matrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{matrix} & \begin{matrix} \begin{pmatrix} & & \\ & M_{\phi_2} & \\ & & \end{pmatrix} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \end{matrix} & 1 \end{matrix} \end{pmatrix}$$

The array M_ϕ is an operator on the wires it intersects. To compute the value of ϕ, it must be applied to a set of wires in which only the first (or top) row R_1 is in state $r_1 = 1$. That is, there is a flow of current along that wire.

We prove that given a crossbar in the state just described, which is induced by the formula ϕ, there exists a path that takes current from the first (or top) row R_1 to the last (or bottom) row $R_{|R|}$ if and only if $\phi = 1$.

Theorem 4. CROSSBAR COMPUTATION OF NNFS *Let $\mathcal{X} = (M, R, C)$ be a crossbar mapping $\phi \in$ NNF. If $r_1 = 1$, then a sneak current will reach $R_{|R|} \in R$ iff $\phi = 1$.*

Proof. The proof is by structural induction on the negation normal form ϕ. We prove that given a

crossbar state where $r_1 = 1$, a crossbar configuration M_ϕ creates a path that takes the wire $R_{|R|}$ to have flow if and only if $\phi = 1$.

Base case: literal. $\phi \equiv a$. Note that $((r_1 \wedge m_{11}) \implies c_1) \wedge (c_1 \wedge m_{21} \implies r_2)$. Thus, for $m_{11} = a = 1$, we have $a \implies r_2$. If $m_{11} = a = 0$, there is no path from R_1 to R_2 s.t. $r_2 = 1$.

Inductive step: conjunction. $\phi \equiv \phi_1 \wedge \phi_2$. We want to prove that $\phi_1 \wedge \phi_2 \iff r_{|R^1|+|R^2|-1}$. Note that M_{ϕ_1} and M_{ϕ_2} share wire $R_{|R^1|}$, which is the output wire of M_{ϕ_1}. Thus, from the inductive hypothesis we have $r_{|R^1|} \iff \phi_1$. $R_{|R^1|}$ is also the input wire to M_{ϕ_2}. It follows that $r_{|R^1|+|R^2|-1} \iff r_{|R^1|} \wedge \phi_2 \iff \phi_1 \wedge \phi_2$.

Inductive step: disjunction. $\phi \equiv \phi_1 \vee \phi_2$. We want to prove that $\phi_1 \vee \phi_2 \iff r_{|R^1|+|R^2|}$.

($\implies$): Suppose $\phi_1 = 1$. From the inductive hypothesis, we have $r_{|R^1|} = 1$. The implications $(r_{|R^1|} \wedge m_{|R^1|,|C^1|+|C^2|+2} \implies c_{|C^1|+|C^2|+2})$, $(c_{|C^1|+|C^2|+2} \wedge m_{|R^1|+|R^2|,|C^1|+|C^2|+2} \implies r_{|R^1|+|R^2|})$ hold. Therefore, $\phi_1 \implies r_{|R^1|+|R^2|}$. Now, suppose $\phi_2 = 1$. A similar chain of implications holds. Namely, it follows that $(r_1 \wedge m_{11} \implies c_1) \wedge (c_1 \wedge m_{|R^1|+1} \implies r_{|R^1|+1})$. Since $R_{|R^1|+1}$ is the input wire to M_{ϕ_2}, it follows that $\phi_2 \implies r_{|R^1|+|R^2|}$.

($\impliedby$): We prove the contrapositive. That is, we prove that $\neg\phi_1 \wedge \neg\phi_2 \implies \neg r_{|R^1|+|R^2|}$. We know from the preceding argument and the inductive hypothesis that $r_{|R^1|} \iff \phi_1$ and $\phi_2 \implies r_{|R^1|+|R^2|}$. Therefore, we must show that $r_{|R^1|+|R^2|} \implies \phi_2$ when $\phi_1 = 0$, which follows from the inductive hypothesis. Thus, we have $r_{|R^1|+|R^2|} = 0$. □

Some examples of Boolean formulas mapped using this inductive design can be seen in Figure 4.3. For the case of a 1-bit adder, the sum bit S can be expressed by the formula $S = (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$. Its crossbar design can be seen in Figure 4.4.

This construction suffices to evaluate Boolean formula. However, in the spirit of parsimony, we

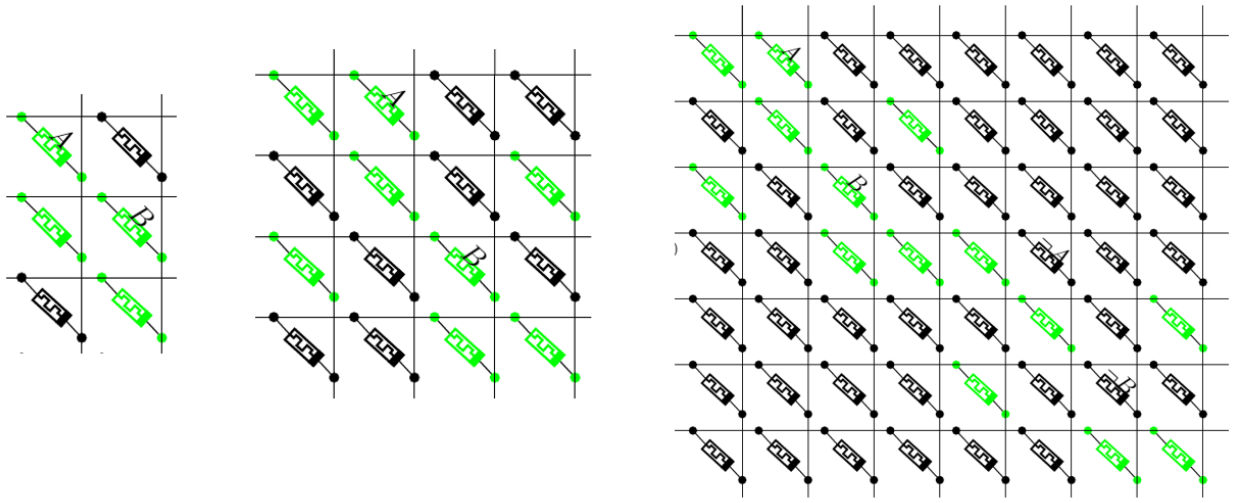


Figure 4.3: Memristor crossbars using our NNF construction for Boolean formulas $(A \wedge B)$ (left), $(A \vee B)$ (center), $A \oplus B \equiv (A \vee B) \wedge (\neg A \vee \neg B)$ (right), where $A = B = 1$ and the source and destination wires are R_1 and $R_{|R|}$, respectively.

seek more space-efficient designs in order to avoid the signal degradation problems that can arise in large crossbars.

It is easily seen that much of the crossbar consists of memristors in the HRS state – this is required to prevent current flow through undesired sneak paths. Similarly, the placement of LRS memristors advances the current of interest such that a current will flow into wire $R_{|R|}$ iff the formula ϕ being mapped to the crossbar evaluates to 1 and an initial current flow is applied at the top row. That is, we assume that $r_1 = 1$.

This method can evaluate Boolean formulas and manages the sneak paths problem without significant added structural complexity to the crossbar. It also does not require a complicated reduction to other logics. However, there are two glaring drawbacks. *First*, the space complexity of the design is very large. *Second*, the poor spatial efficiency can lead to poor delay and power consumption due to the overhead of configuring such a large crossbar to the desired state.

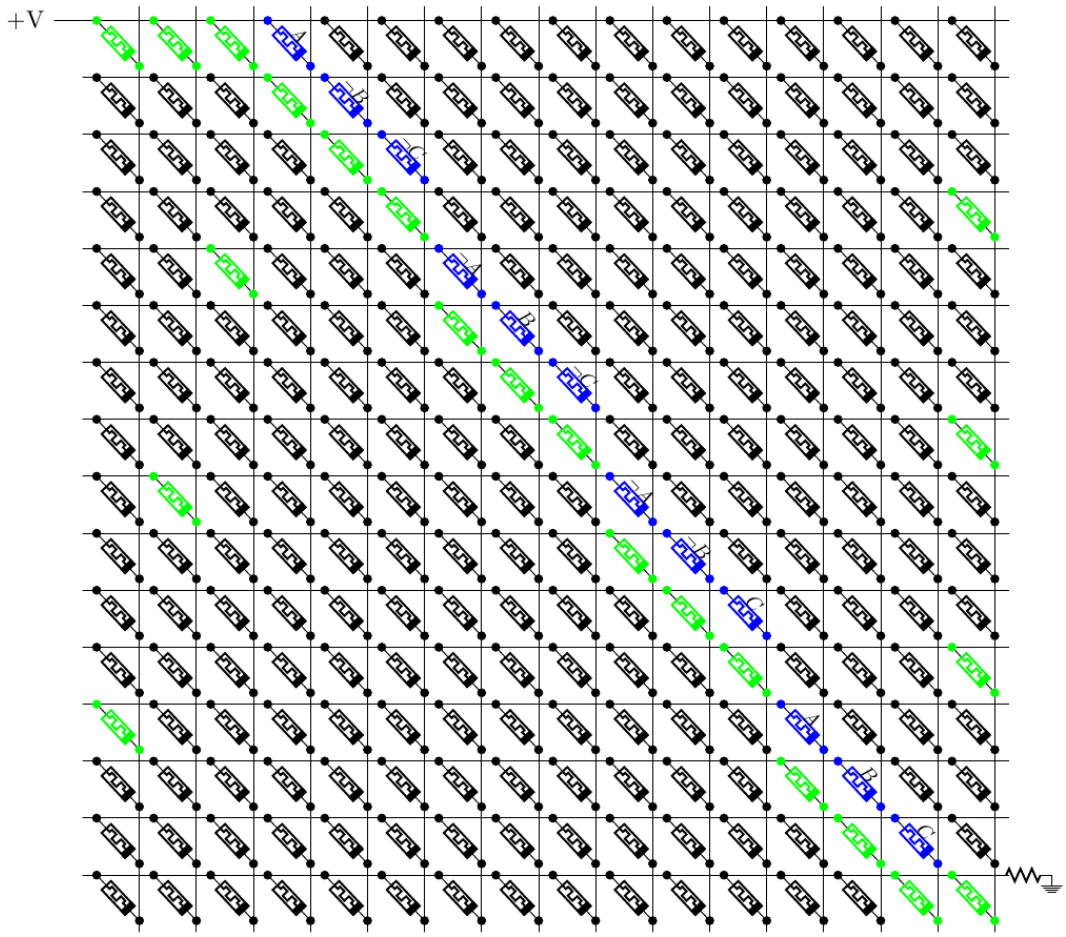


Figure 4.4: Crossbar mapping the sum bit $S = (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$. A voltage is applied to wire R_1 and R_{16} is grounded in order to read the value corresponding to S. Blue memristors map a Boolean literal from ϕ. Green memristors are in the LRS state, black memristors are in the HRS state, and the states of blue memristors are determined by variables A, B, C and their negations. Simulation files can be found in eecs.ucf.edu/∼velasquez/Homogeneous.

CHAPTER 5: DESIGN AUTOMATION

While a universal design procedure was presented in the previous chapter, the resulting crossbars were very large. In this chapter, we present a framework for generating compact crossbar designs to compute a given Boolean formula. Our synthesis procedure is particularly well-suited for mitigating common types of faults at the post-fabrication stage of the design process. We will demonstrate how our approach can still provide valid crossbar configurations in the presence of highly defective devices. The following faults are accounted for:

- Stuck-off: A component M_{ij} has a stuck-off fault if $m_{ij} = 0$ and the node cannot be programmed to another state. That is, the interconnect is permanently in the HRS state.
- Stuck-on: A component M_{ij} has a stuck-on fault if $m_{ij} = 1$ and the node cannot be programmed to another state. That is, the interconnect is permanently in the LRS state.
- Broken wire: This fault corresponds to wires that are segmented into multiple pieces. The affected write R_i is represented by its constituent segments, where $r_{i,k}$ denotes the value of the k^{th} segment of R_i.

Our approach utilizes the paths of current throughout the crossbar in order to compute a Boolean formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$, where $\mathbb{B}$ is the set $\{0, 1\}$. As we have seen, by methodically programming the crossbar components to be variables in ϕ, we can manipulate the trajectories of paths induced so that there is a flow of current between two specified wires if and only if ϕ holds. Given $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$, we represent the set version of the k^{th} formula ϕ^k as Φ^k, where Φ^k_i denotes the i^{th} clause of Φ^k and Φ^k_{ij} denotes the j^{th} variable in the i^{th} clause. For example, given $\phi : \mathbb{B}^2 \mapsto \mathbb{B}^2$, where $\phi^1 = (a \wedge b) \vee (\neg a \wedge \neg b)$ and $\phi^2 = (\neg a \wedge b) \vee (a \wedge \neg b)$, we represent these formu-

las as sets $\Phi^1 = \{\{a, b\}, \{\neg a, \neg b\}\}$ and $\Phi^2 = \{\{\neg a, b\}, \{a, \neg b\}\}$, respectively. This alternative representation facilitates later proofs.

Thus far, we have been concerned with traditional crossbars consisting of two-terminal components capable of transmitting current in both directions. From this point on, we will refer to such crossbars as homogeneous. We have formally defined homogeneous crossbars in the previous chapter. For convenience, we provide a short definition below.

Definition 4 (Homogeneous Crossbar)**.** *A homogeneous crossbar is a tuple $\mathcal{X} = (M, R, C)$, where $M = (M_{ij})$ is the set of components such that $m_{ij} = 1$ ($m_{ij} = 0$) denotes an LRS (HRS) node. The sets $R = (R_i)$ and $C = (C_j)$ denote the sets of wordlines, or row wires, and bitlines, or column wires. A value of $r_i = 1$ ($r_i = 0$) denotes the presence (absence) of electric current in wire R_i. For convenience, we often deal with the general wire set $W = R \cup C$.*

In a crossbar $\mathcal{X} = (M, R, C)$, if M_{ij} is in the LRS state ($m_{ij} = 1$), it redirects current from one terminal to the other. This motivates Axiom 2.

Axiom 2 (Homogeneous Flow)**.** *Given a homogeneous crossbar $\mathcal{X} = (M, R, C)$, $(r_i \wedge m_{ij}) \implies c_j$ and $(c_j \wedge m_{ij}) \implies r_i$ always hold. As a result, property (5.1) always holds.*

$$\bigwedge_{j=1}^{|C|} \left(c_j \iff \bigvee_{i=1}^{|R|} m_{ij} \wedge r_i \right) \wedge \bigwedge_{i=1}^{|R|} \left(r_i \iff \bigvee_{j=1}^{|C|} m_{ij} \wedge c_j \right) \tag{5.1}$$

As its name suggests, paths-based logic seeks to use the paths of current throughout the crossbar as a means of computation. An initial flow of current is generated by applying a voltage bias to some source wire and grounding another wire. As we have seen earlier, a path is a sequence of nodes connecting two wires. For example, the path $\Pi^{R_i \to C_j} = (M_{ij_1}, M_{i_1 j_1}, M_{i_1 j_2}, M_{i_2 j_2}, \ldots, M_{i_k j})$ connects wires R_i and C_j, where $\pi_d^{R_i \to C_j}$ denotes the value of the d^{th} component. From Axiom 1, the

following chain of implications holds.

$$\begin{aligned}(r_i \wedge m_{ij_1} \implies c_{j_1}) \wedge (c_{j_1} \wedge m_{i_1 j_1} \implies r_{i_1}) \wedge \\ (r_{i_1} \wedge m_{i_1 j_2} \implies c_{j_2}) \wedge \cdots \wedge (r_{i_k} \wedge m_{i_k j} \implies c_j)\end{aligned} \tag{5.2}$$

We can also show that $(c_j \wedge m_{i_k j} \implies r_{i_k}) \wedge \cdots \wedge (c_{j_1} \wedge m_{ij_1} \implies r_i)$. Thus, if every component in a path $\Pi^{W_i \to W_j}$ is in the LRS state, then current flowing in the source wire W_i will be redirected to the destination wire W_j, and vice-versa. We call this the symmetry of paths property. A general form of this property is captured by equation (5.3).

$$\left[w_i \wedge \bigwedge_d \pi_d^{W_i \to W_j} \implies w_j \right] \wedge \left[w_j \wedge \bigwedge_d \pi_d^{W_i \to W_j} \implies w_i \right] \tag{5.3}$$

We have seen that paths can be treated as a conjunction of variables mapped to components. Thus, it is convenient to think of formulas in their disjunctive normal form (DNF). Any Boolean formula can be written in DNF as a disjunction of conjunctive clauses (i.e. $\bigvee_i \bigwedge_j \phi_{ij}$). For the remainder of this thesis, we assume that all Boolean formulas are in DNF. We do this for ease of presentation and it is not a restriction on the methods proposed.

Given a formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and a crossbar $\mathcal{X} = (M, R, C)$, we want to find a mapping of the components M to variables $b_1, \ldots, b_p$ in ϕ and constants $\{0, 1\}$ denoting the HRS and LRS states. We define such a mapping by $P = (P_{ij})$, where $P_{ij} \in \{B_1, \neg B_1, B_2, \neg B_2, \ldots, B_p, \neg B_p\} \cup \{0, 1\}$. P defines a configuration of M such that (5.4) holds. See Fig. 5.1 for an example.

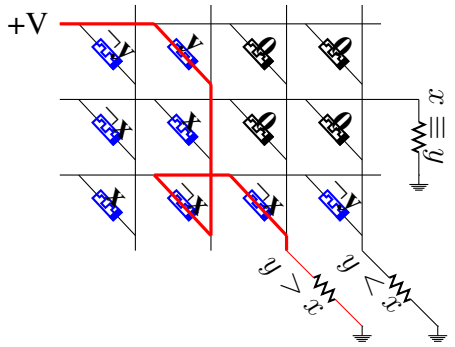
$$\begin{aligned}(m_{ij} = 1) \iff (P_{ij} = 1) \vee ((P_{ij} = B_k) \wedge b_k) \vee \\ ((P_{ij} = \neg B_k) \wedge \neg b_k)\end{aligned} \tag{5.4}$$

Definition 5 (Well-Formed Design)**.** *Suppose we are given a Boolean formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and crossbar $\mathcal{X} = (M, R, C)$ with input and output wires $S, F \subseteq W$ such that the value of the k^{th} formula ϕ^k will be output to wire F_k. A well-formed design is a mapping matrix P such that there are paths $\Pi^{S_i \to F_j}$ that satisfy (5.5) for all evaluations $\alpha \in \mathbb{B}^p$ of ϕ.*

$$\bigwedge_{k=1}^{q} \left(f_k \iff \phi^k\right) \tag{5.5}$$

In order for (5.5) to hold, there must be paths from the source wires S to the destination wires F such that current will flow from source S_i to destination F_j if and only if ϕ^j is true. Let us elucidate this with an example. Suppose we want to find a well-formed design for a 1-bit comparator using a 3×4 homogeneous crossbar $\mathcal{X} = (M, (R_1, R_2, R_3), (C_1, C_2, C_3, C_4))$. Given two bits x and y, we want outputs indicating the three possible outcomes $(x \equiv y) = (x \wedge y) \vee (\neg x \wedge \neg y)$, $(y > x) = \neg x \wedge y$, or $(y < x) = x \wedge \neg y$. Let $S = (R_1)$ and $F = (R_2, C_3, C_4)$ denote the sets of source and destination wires, respectively. Suppose $s_1 = 1$ for all evaluations $\alpha \in \mathbb{B}^2$. This means that there will be a flow of current on S_1 regardless of the values in the evaluation vector. We want a well-formed design P that assigns values to each component M_{ij} so that there will be paths $\Pi^{R_1 \to R_2}$, $\Pi^{R_1 \to C_3}$, $\Pi^{R_1 \to C_4}$ from the source wire to the destination wires resulting in $r_2 \iff (x \equiv y)$, $c_3 \iff (y > x)$, and $c_4 \iff (y < x)$. For each evaluation $\alpha \in \mathbb{B}^2$, the paths are as follows. See Fig. 5.1 for a visual interpretation.

- $\alpha = (x = 0, y = 0)$: $\Pi^{R_1 \to R_2} = (M_{11}, M_{21})$
- $\alpha = (x = 0, y = 1)$: $\Pi^{R_1 \to C_3} = (M_{12}, M_{32}, M_{33})$
- $\alpha = (x = 1, y = 0)$: $\Pi^{R_1 \to C_4} = (M_{11}, M_{31}, M_{34})$
- $\alpha = (x = 1, y = 1)$: $\Pi^{R_1 \to R_2} = (M_{12}, M_{22})$



$$P = \begin{array}{c} \\ R_1 \\ R_2 \\ R_3 \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} C_1 & C_2 & C_3 & C_4 \end{array} \\ \begin{pmatrix} \neg y & y & 0 & 0 \\ \neg x & x & 0 & 0 \\ x & \neg x & \neg x & \neg y \end{pmatrix} \end{array} \tag{5.6}$$

Figure 5.1: Crossbar $\mathcal{X} = (M, R = (R_1, R_2, R_3), C = (C_1, C_2, C_3, C_4))$ with mapping matrix (5.6). If a voltage pulse is applied to R_1 and R_2, C_3, C_4 are grounded, then we have an initial flow of current $r_1 = 1$. The red bars represent the current flow from R_1 to C_3 when $y > x$, i.e. when $y \wedge \neg x$ holds.

Bounded Model Checking

We pose the problem of finding a well-formed design given the foregoing crossbar dynamics as a model checking problem. Bounded model checking (BMC) algorithms construct a formula $\mathcal{M}_{\text{BMC}}$ (5.7) from a state-transition system based on an initial state I, a transition relation τ, a property specification to be checked ψ, and a time bound T specifying the maximum length of a trajectory in the state-transition graph [77]. If a state is found wherein $\mathcal{M}_{\text{BMC}}$ is evaluated to be false, a counterexample is produced. We will demonstrate how to build a state-transition system such that said counterexample produces a well-formed design P that computes a given formula ϕ.

Given $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and a candidate design P for some crossbar $\mathcal{X} = (M, R, C)$ with source and destination wires S, F, we have 2^p evaluation vectors $\alpha \in \mathbb{B}^p$. For each such α, we define a finite state machine $\mathcal{L}_\alpha$, where $\mathcal{L}_\alpha[m_{ij}]$, $\mathcal{L}_\alpha[r_i]$, and $\mathcal{L}_\alpha[c_j]$ denote the values of m_{ij}, r_i, and c_j induced by design P under evaluation α. Each $\mathcal{L}_\alpha$ is a component in a transition system $\mathcal{L}$ consisting of an initial state (5.8) and a transition relation (5.9) defining the dynamics of the system as it moves from state u_t to state u_{t+1} (See Fig. 5.2). The initial state specifies which wire(s) have an initial

flow of current and what values the components in the memory are initialized to. This information is given by the source wire set S and the candidate design P. The transition relation encodes Axiom 2. We are looking for a state u_t such that specification (5.10) is violated. We have defined (5.10) as the negation of (5.5) so that a counterexample to (5.10) yields a well-formed design P, i.e. one where (5.5) holds.

In order to determine the value of bound T, recall the results established in the previous chapter. Namely, that the set of paths of length at most $2(\min\{|R|, |C|\})$ between any two wires has the same computing power under paths-based logic as the set of all paths between said wires [78]. Using Fig. 5.2 as a reference, note that each transition from state u_t to u_{t+1} corresponds to some M_{ij} redirecting current from one of its terminals to the other, where u_0 is the initial state. Thus, we need only look at trajectories in $\mathcal{L}$ of length at most $T = 2(\min\{|R|, |C|\})$ to determine whether a counterexample to (5.10) exists.

$$\mathcal{M}_{\text{BMC}} \triangleq I(u_0) \wedge \bigwedge_{i=0}^{T-1} \tau(u_i, u_{i+1}) \wedge \bigwedge_{i=0}^{T} \psi(u_i) \tag{5.7}$$

$$\begin{aligned} I(u_0) \triangleq \bigwedge_{\alpha \in \mathbb{B}^p} \left(\bigwedge_{w_i \in S} (\mathcal{L}_\alpha^0[w_i] \iff s_i) \right) \wedge \left(\bigwedge_{w_i \notin S} \neg \mathcal{L}_\alpha^0[w_i] \right) \wedge \\ \left(\bigwedge_{i,j,k} \mathcal{L}_\alpha[m_{ij}] \iff (p_{ij} = 1) \vee ((p_{ij} = B_k) \wedge \mathcal{L}_\alpha[b_k]) \vee ((p_{ij} = \neg B_k) \wedge \neg \mathcal{L}_\alpha[b_k]) \right) \end{aligned} \tag{5.8}$$

$$\begin{aligned} \tau(u_t, u_{t+1}) \triangleq \bigwedge_{\alpha \in \mathbb{B}^p} \left(\bigwedge_{i=1}^{|R|} \left(\mathcal{L}_\alpha^{t+1}[r_i] \iff \bigvee_{j=1}^{|C|} \mathcal{L}_\alpha[m_{ij}] \wedge \mathcal{L}_\alpha^t[c_j] \right) \wedge \right. \\ \left. \bigwedge_{j=1}^{|C|} \left(\mathcal{L}_\alpha^{t+1}[c_j] \iff \bigvee_{i=1}^{|R|} \mathcal{L}_\alpha[m_{ij}] \wedge \mathcal{L}_\alpha^t[r_i] \right) \right) \end{aligned} \tag{5.9}$$

$$\psi(u_t) \triangleq \neg \left(\bigwedge_{\alpha \in \mathbb{B}^p} \left(\bigwedge_{k=1}^{q} \left(\mathcal{L}_\alpha[f_k] \iff \phi^k \right) \right) \right) \tag{5.10}$$

As an example, suppose we are given $\phi = (a \wedge \neg b) \vee (\neg a \wedge b)$ and a crossbar $\mathcal{X} = ((M, R = (R_1, R_2), C = (C_1, C_2))$ with input/output wires $S = (R_1), F = (R_2)$ such that $s_1 = 1$ for all evaluations of a, b, c. Suppose the bounded model checking procedure is verifying whether the design P is a well-formed design, where $p_{11} = A$, $p_{12} = \neg A$, $p_{21} = B$, $p_{22} = \neg B$. Then the formulas (5.8), (5.9), (5.10) for this problem are (5.11), (5.12), (5.13), respectively.

$$\begin{aligned}
I(u_0) \triangleq & \left(\bigwedge_{\alpha \in \mathbb{B}^2} \mathcal{L}^0_\alpha[r_1] \wedge \neg \mathcal{L}^0_\alpha[r_2] \wedge \neg \mathcal{L}^0_\alpha[c_1] \wedge \neg \mathcal{L}^0_\alpha[c_2] \right) \wedge \\
& \bigwedge_{\alpha \in \mathbb{B}^2} \mathcal{L}_\alpha[m_{11}] \iff (p_{11} = 1) \vee ((p_{11} = A) \wedge \mathcal{L}_\alpha[a]) \vee ((p_{11} = \neg A) \wedge \neg \mathcal{L}_\alpha[a]) \vee \\
& \qquad ((p_{11} = B) \wedge \mathcal{L}_\alpha[b]) \vee ((p_{11} = \neg B) \wedge \neg \mathcal{L}_\alpha[b]) \wedge \\
& \bigwedge_{\alpha \in \mathbb{B}^2} \mathcal{L}_\alpha[m_{12}] \iff (p_{12} = 1) \vee ((p_{12} = A) \wedge \mathcal{L}_\alpha[a]) \vee ((p_{12} = \neg A) \wedge \neg \mathcal{L}_\alpha[a]) \vee \\
& \qquad ((p_{12} = B) \wedge \mathcal{L}_\alpha[b]) \vee ((p_{12} = \neg B) \wedge \neg \mathcal{L}_\alpha[b]) \wedge \\
& \bigwedge_{\alpha \in \mathbb{B}^2} \mathcal{L}_\alpha[m_{21}] \iff (p_{21} = 1) \vee ((p_{21} = A) \wedge \mathcal{L}_\alpha[a]) \vee ((p_{21} = \neg A) \wedge \neg \mathcal{L}_\alpha[a]) \vee \\
& \qquad ((p_{21} = B) \wedge \mathcal{L}_\alpha[b]) \vee ((p_{21} = \neg B) \wedge \neg \mathcal{L}_\alpha[b]) \wedge \\
& \bigwedge_{\alpha \in \mathbb{B}^2} \mathcal{L}_\alpha[m_{22}] \iff (p_{22} = 1) \vee ((p_{22} = A) \wedge \mathcal{L}_\alpha[a]) \vee ((p_{22} = \neg A) \wedge \neg \mathcal{L}_\alpha[a]) \vee \\
& \qquad ((p_{22} = B) \wedge \mathcal{L}_\alpha[b]) \vee ((p_{22} = \neg B) \wedge \neg \mathcal{L}_\alpha[b])
\end{aligned} \tag{5.11}$$

$$
\begin{aligned}
\tau(u_0,u_1) \triangleq & \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^1_\alpha[r_1] \iff (\mathcal{L}^0_\alpha[r_1] \vee (\mathcal{L}^0_\alpha[c_1] \wedge \mathcal{L}_\alpha[m_{11}]) \vee (\mathcal{L}^0_\alpha[c_2] \wedge \mathcal{L}_\alpha[m_{12}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^1_\alpha[r_2] \iff (\mathcal{L}^0_\alpha[r_2] \vee (\mathcal{L}^0_\alpha[c_1] \wedge \mathcal{L}_\alpha[m_{21}]) \vee (\mathcal{L}^0_\alpha[c_2] \wedge \mathcal{L}_\alpha[m_{22}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^1_\alpha[c_1] \iff (\mathcal{L}^0_\alpha[c_1] \vee (\mathcal{L}^0_\alpha[r_1] \wedge \mathcal{L}_\alpha[m_{11}]) \vee (\mathcal{L}^0_\alpha[r_2] \wedge \mathcal{L}_\alpha[m_{12}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^1_\alpha[c_2] \iff (\mathcal{L}^0_\alpha[c_2] \vee (\mathcal{L}^0_\alpha[r_1] \wedge \mathcal{L}_\alpha[m_{12}]) \vee (\mathcal{L}^0_\alpha[r_2] \wedge \mathcal{L}_\alpha[m_{22}]))\right) \\
\tau(u_1,u_2) \triangleq & \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^2_\alpha[r_1] \iff (\mathcal{L}^1_\alpha[r_1] \vee (\mathcal{L}^1_\alpha[c_1] \wedge \mathcal{L}_\alpha[m_{11}]) \vee (\mathcal{L}^1_\alpha[c_2] \wedge \mathcal{L}_\alpha[m_{12}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^2_\alpha[r_2] \iff (\mathcal{L}^1_\alpha[r_2] \vee (\mathcal{L}^1_\alpha[c_1] \wedge \mathcal{L}_\alpha[m_{21}]) \vee (\mathcal{L}^1_\alpha[c_2] \wedge \mathcal{L}_\alpha[m_{22}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^2_\alpha[c_1] \iff (\mathcal{L}^1_\alpha[c_1] \vee (\mathcal{L}^1_\alpha[r_1] \wedge \mathcal{L}_\alpha[m_{11}]) \vee (\mathcal{L}^1_\alpha[r_2] \wedge \mathcal{L}_\alpha[m_{12}]))\right) \wedge \\
& \bigwedge_{\alpha\in\mathbb{B}^2} \left(\mathcal{L}^2_\alpha[c_2] \iff (\mathcal{L}^1_\alpha[c_2] \vee (\mathcal{L}^1_\alpha[r_1] \wedge \mathcal{L}_\alpha[m_{12}]) \vee (\mathcal{L}^1_\alpha[r_2] \wedge \mathcal{L}_\alpha[m_{22}]))\right)
\end{aligned}
\tag{5.12}
$$

$$
\begin{aligned}
\psi(u_0) &\triangleq \left(\bigwedge_{\alpha\in\mathbb{B}^2} \mathcal{L}^0_\alpha[r_2] \iff \phi\right) \\
\psi(u_1) &\triangleq \left(\bigwedge_{\alpha\in\mathbb{B}^2} \mathcal{L}^1_\alpha[r_2] \iff \phi\right) \\
\psi(u_2) &\triangleq \left(\bigwedge_{\alpha\in\mathbb{B}^2} \mathcal{L}^2_\alpha[r_2] \iff \phi\right)
\end{aligned}
\tag{5.13}
$$

A visualization of the state-transition system induced by the model checker for the comparator in Fig. 5.1 can be seen in Fig. 5.2. A counterexample to (5.10) is found in state u_3. Indeed, assuming a source wire R_1 and destination wires R_2, C_3, C_4, note that we have $\mathcal{L}^3_\alpha[r_2] \iff (x \equiv y)$, $\mathcal{L}^3_\alpha[c_3] \iff (y > x)$, and $\mathcal{L}^3_\alpha[c_4] \iff (y < x)$, thereby violating specification (5.10).

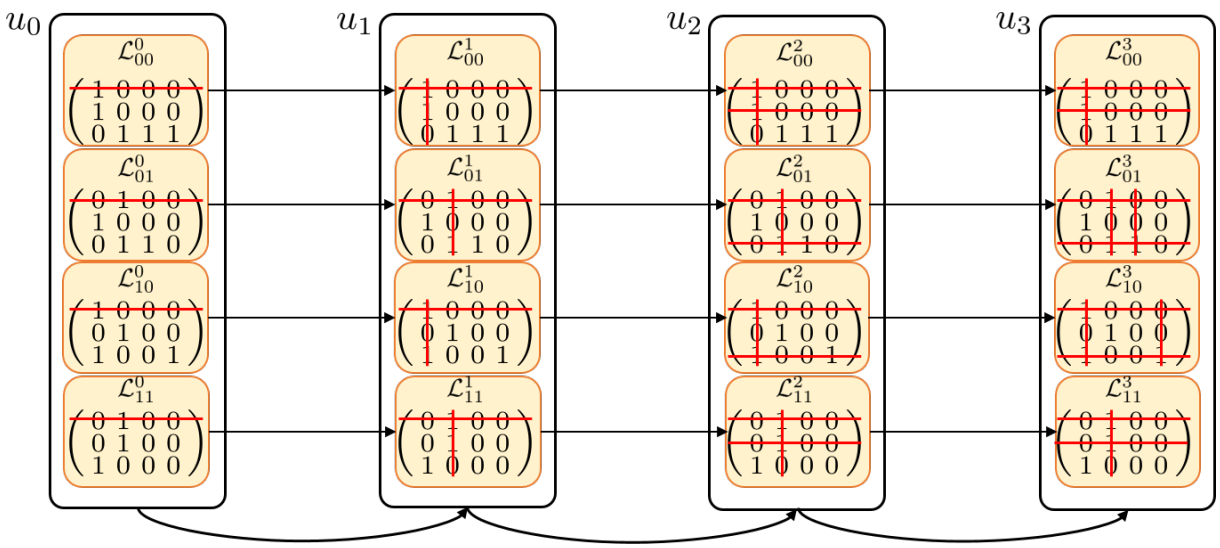


Figure 5.2: The finite state machine $\mathcal{L}$ given the design P (5.6) for the comparator in Fig. 5.1. There are 4 sub-automata $\mathcal{L}_{00}, \mathcal{L}_{01}, \mathcal{L}_{10}, \mathcal{L}_{11}$ corresponding to each evaluation vector $\alpha \in \mathbb{B}^2$. Each state u_t in $\mathcal{L}$ is the union of states of its sub-automata $\mathcal{L}^t_\alpha$.

Experimental Results

We use our approach to automatically generate crossbar designs for the following formulae (design and circuit generation files can be found in eecs.ucf.edu/∼velasquez/Homogeneous.).

- 3-bit parity: $\phi = (\neg b_1 \wedge \neg b_2 \wedge b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (b_1 \wedge b_2 \wedge b_3)$ is mapped to a 3×3 crossbar with source/destination wires $S = (R_3)$, $F = (R_1)$.

- 4-bit parity: $\phi = (\neg b_1 \vee \neg b_2 \vee b_3 \vee b_4) \wedge (\neg b_1 \vee b_2 \vee \neg b_3 \vee b_4) \wedge (\neg b_1 \vee b_2 \vee b_3 \vee \neg b_4) \wedge (b_1 \vee \neg b_2 \vee \neg b_3 \vee b_4) \wedge (b_1 \vee \neg b_2 \vee b_3 \vee \neg b_4) \wedge (b_1 \vee b_2 \vee \neg b_3 \vee \neg b_4) \wedge (b_1 \vee b_2 \vee b_3 \vee b_4) \wedge (\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee \neg b_4)$ is mapped to a 3×4 crossbar with $S = (R_3)$, $F = (R_1)$.

- Full adder: This design consists of computing the sum and carry-out bits simultaneously within a single crossbar. We do so by mapping $\phi^1 = Sum = (\neg b_1 \wedge \neg b_2 \wedge b_3) \vee (\neg b_1 \wedge b_2 \wedge$

Table 5.1: Comparison of design performance, where time is defined as the number of steps required to configure the crossbar and compute the formula of interest.

	2-bit XOR		1-bit Sum		4-bit Parity	
	Size	Time	Size	Time	Size	Time
[1]	5×4	4 steps	7×6	6 steps	9×10	10 steps
[2]	3×3	7 steps	5×4	9 steps	9×5	11 steps
[76]	7×8	8 steps	16×16	17 steps	40×40	41 steps
[3]	2×2	3 steps	3×8	7 steps	3×31	7 steps
Auto. Synth.	2×2	3 steps	3×3	4 steps	3×4	4 steps

$\neg b_3) \vee (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (b_1 \wedge b_2 \wedge b_3)$ and $\phi^2 = C_{out} = (b_1 \wedge b_2) \vee (b_1 \wedge b_3) \vee (b_2 \wedge b_3)$ to a 4×5 crossbar with source/destination wires $S = (R_4)$, $F = (R_1, R_2)$.

After reviewing the memristor crossbar generated in Fig. 5.4, we can compare this with our manual design for the sum bit of a full adder (Fig. 4.4) in Chapter 4. The space efficiency of the automatically generated design is due to the use of literals mapped to memristors in such a way that the literal values themselves prevent unwanted sneak paths – hence drastically reducing the number of HRS memristors required just for this purpose. These more space-efficient crossbars inherently have the attributes of faster computation time due to reduced crossbar-configuration overhead, and higher cost-effectiveness. See Table 5.1 for a comparison.

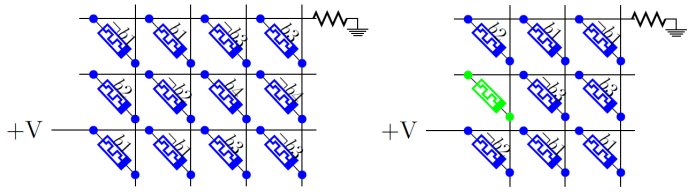


Figure 5.3: Crossbar design visualization of a 4-bit parity and the sum bit of a full adder.

We pay particular attention to the full adder due to its tremendous significance in modern computing. In order to study the performance of our full adder designs experimentally, crossbar arrays

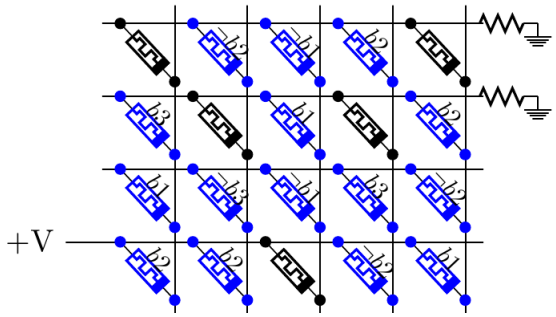


Figure 5.4: Crossbar design visualization of a full adder.

(12×12) were implemented on a 300 mm wafer platform in a back-end-of-the-line (BEOL) process. The IBM 65 nm 10LPe process was used as the mainframe to integrate a custom build ReRAM layer between metal 1 (M1) and metal 2 (M2) (See Fig. 5.5). Metal 1 was fabricated in a damascene process where 100 nm lines were etched into the SiO2 insulating layer and subsequently filled with a line consisting of TaN/Ta followed by electroplated Cu. After a planarization step, the HfOx layer was deposited in a reactive PVD process and subsequently served as the active part of the memristor. A subsequent TiN deposition created the top electrode and in a RIE step followed by a wet etch, the 12×12 memristor elements were defined in the size of 100×100 nm^2 on top of M1. By depositing SiO2 the active layer was encapsulated and the M2 insulation was fabricated. A final dual-damascene step created the 100 nm wide M2 trenches and V1 holes to connect to the created TiN top electrode. M2/V1 got fabricated by putting down the liner (TaN/Ta) and again electroplating Cu. A final planarization step removed the excess copper resulting in insulated M2 lines. For our adder circuit implementation, we used the 4×5 sub-matrix in a corner of our 12×12 matrix array. More details can be found in [79].

The adder operation was performed in two steps. In the first step, the array was configured to one of the input combinations. In a commercial realization, a control circuit might be used to change one input state to another input state. In the second step, pulse voltages of 0.1 and 0.2 volts were

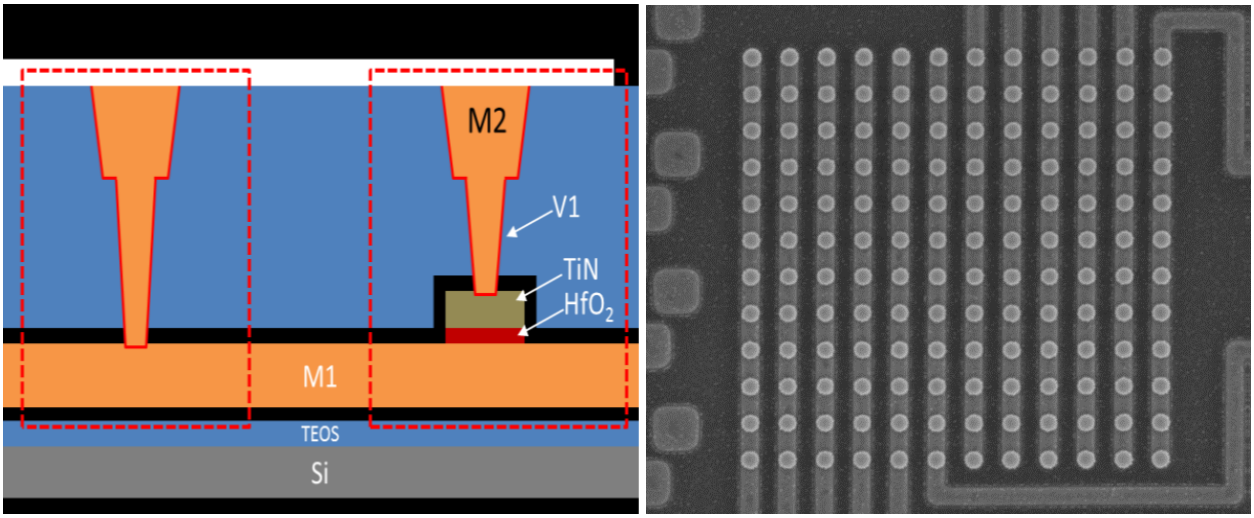


Figure 5.5: (left) Schematic cross-section of a memristor used in this work. (right) SEM View of a 12×12 memristor crossbar array. Crossbar arrays were fabricated at SUNY Polytechnic Institute on 300 mm wafers using a modified IBM 65 nm 10LPe process flow.

applied to wire R_4 and the voltages across wires R_1 and R_2 for all the input configurations of the 1-bit full adder were measured. In all the cases, output voltages corresponded to the expected logic levels. The output voltage for output logic level 1 was at least 20 times higher than that of output logic level 0. For example, when the inputs are $A = 1, B = 1, C = 0$, the output logic levels are: $Sum = 0, C_{out} = 1$. In this case, the values $V(R_1)$ = 1.77 mv and $V(R_2)$ = 98.04 mV were recorded. In this case, $V(R_2)/V(R_1)$ = 55. The experimental results are summarized in the Table 5.2.

The speed of computation depends on the delay between the input and output signals. Cadence Spectra circuit simulator (SPICE level simulator), tightly integrated with virtuoso custom design platform, was used to design a schematic of a 4×5 crossbar array and a traditional 28T CMOS adder and to simulate propagation delay and other metrics for both types of adders. Our crossbar array was fabricated on the IBM 65 nm CMOS 10LPe Low Power technology platform. Line resistances and other parasitic capacitances such as lateral capacitance, fringe capacitance etc. were

extracted for simulation following IBM 10LPe design specifications. The LRS state memristor resistance values were set to 1 kΩ, and the HRS memristor resistances were set to 1 MΩ. The 28T adder was designed based on the same technology platform and optimal transistor size was used to maintain optimal delay.

Table 5.2: Output voltages for wires R_1 and R_2 in the full adder design in Figure 5.4 under all possible evaluations.

b_1	b_2	b_3	R_1 Voltage	R_2 Voltage	Sum	C_{out}
0	0	0	0.94	1.33	0	0
0	0	1	95.53	0.74	1	0
0	1	0	98.50	0.79	1	0
0	1	1	1.04	98.43	0	1
1	0	0	98.95	0.84	1	0
1	0	1	4.58	98.42	0	1
1	1	0	1.77	98.04	0	1
1	1	1	97	99.04	1	1

Table 5.3: Comparison between the proposed full adder architecture and the traditional 28-transistor CMOS full adder.

	28T (1 GHz)	Crossbar (1 Volt input)	Crossbar (0.2 Volt input)
Propagation Delay (ps)	39	1.4	1.3
Power (μ W)	4.68	22.5	1
PDP (x1E-17 Joules)	182.52	31.5	1.3

Fault Tolerance

When compared to lithographic CMOS device manufacturing, self-assembled nanodevices have high defect rates in the form of stuck-off and stuck-on devices, as well as broken or partitioned nanowires [80]. This necessitates fault-tolerant mapping schemes at the post-manufacturing stage

[14] since rejecting faulty chips becomes unacceptable due to the high probability of defects per chip [81].

One of the benefits of the proposed approach is how well it lends itself for modeling such crossbar faults. To account for stuck-on and stuck-off defects, we define specification (5.14). Given a crossbar $\mathcal{X} = (M, R, C)$, we define a defect matrix $Z \in \{+, -, *\}^{|R|\times|C|}$, where $z_{ij} = +$ and $z_{ij} = -$ denote stuck-on/-off defects in node M_{ij}.

$$\bigwedge_{i=1}^{|R|}\bigwedge_{j=1}^{|C|}(z_{ij} = + \implies p_{ij} = 1) \wedge (z_{ij} = - \implies p_{ij} = 0) \tag{5.14}$$

Broken wires are also accounted for. Let $\theta^{R_i} = (1, \theta_2^{R_i}, \theta_3^{R_i}, \ldots, |C|+1)$ and $\theta^{C_j} = (1, \theta_2^{C_j}, \theta_3^{C_j}, \ldots, |R|+1)$ be increasing sequences representing the position of breaks in wires R_i and C_j, respectively, such that $k \in \theta^{R_i}$ denotes a break in wire R_i between columns C_{k-1} and C_k, thereby splitting the wire into two segments $R_{i,1}$ and $R_{i,2}$. Similar reasoning applies to $k \in \theta^{C_j}$. A wire R_i with no breaks has $\theta^{R_i} = (1, n+1)$. These ideas are formalized in (5.15) and (5.16), where m_{ij}^r and m_{ij}^c denote the row and column wire segments that M_{ij} resides in. Specification (5.16) replaces the transition relation (5.9) when there are broken wires. By adding specifications (5.14) and (5.15), we can generate designs for faulty crossbars. See Fig. 5.6 for a 4-bit comparator design using highly defective crossbars.

$$\bigwedge_{i=1}^{|R|}\bigwedge_{j=1}^{|C|}\left(\bigwedge_{k=1}^{|\theta^{R_i}|-1}\theta_k^{R_i} \le j < \theta_{k+1}^{R_i} \implies m_{ij}^r = k\right) \wedge \left(\bigwedge_{k=1}^{|\theta^{C_j}|-1}\theta_k^{C_j} \le i < \theta_{k+1}^{C_j} \implies m_{ij}^c = k\right) \tag{5.15}$$

$$\bigwedge_{\alpha\in\mathbb{B}^p}\bigwedge_{i=1}^{|R|}\left(\mathcal{L}_\alpha^{t+1}[r_{i,m_{ij}^c}] \iff \bigvee_{j=1}^{|C|}\left(\mathcal{L}_\alpha[m_{ij}]\wedge\mathcal{L}_\alpha^t[c_{j,m_{ij}^r}]\right)\right)\wedge \bigwedge_{j=1}^{|C|}\left(\mathcal{L}_\alpha^{t+1}[c_{j,m_{ij}^r}] \iff \bigvee_{i=1}^{|R|}\left(\mathcal{L}_\alpha[m_{ij}]\wedge\mathcal{L}_\alpha^t[r_{i,m_{ij}^c}]\right)\right) \tag{5.16}$$

Crossbar Networks

While the bounded model checking approach proposed in this chapter yields very compact and correct crossbar designs, its runtime complexity is a function of the number of paths in the crossbar, which we have shown to be exponential. Naturally, one might think to break the formula into smaller sub-formulas, generate designs for said sub-formulas, and merge the resulting designs. Such a design philosophy is adapted in CMOS to fabricate n-bit adders and multipliers as sequences of 1-bit adders. Indeed, we can apply a similar approach.

Let us consider the problem of generating a crossbar design for an n-bit comparator. Let $x = (x_n, x_{n-1}, \ldots, x_1)$ and $y = (y_n, y_{n-1}, \ldots, y_1)$ denote two n-bit vectors, where x_n and y_n represent the most significant bits (MSBs). There are three outputs that we would like to measure corresponding to the cases where $x \equiv y$, $y > x$, and $y < x$. Let us consider how this operation can be mapped to n inter-connected crossbars. Let $(x \equiv y)_i$ denote whether bit-vectors $(x_n, \ldots, x_i)$ and $(y_n, \ldots, y_i)$ are equivalent. Furthermore, let $(y > x)_i$ and $(y < x)_i$ denote whether the bit-vectors first differ on the i^{th} bit so that $y > x$ and $y < x$ hold, respectively. These equations are defined below.

$$(x \equiv y)_i \iff (x \equiv y)_{i+1} \wedge ((x_i \wedge y_i) \vee (\neg x_n \wedge \neg y_n)), (x \equiv y)_{n+1} = 1 \tag{5.17}$$

$$(y > x)_i \iff (x \equiv y)_{i+1} \wedge (\neg x_i \wedge y_i) \tag{5.18}$$

$$(y < x)_i \iff (x \equiv y)_{i+1} \wedge (x_i \wedge \neg y_i) \tag{5.19}$$

We can define crossbars $\mathcal{X}^n, \ldots, \mathcal{X}^1$, where $\mathcal{X}^i = (M^i, R^i, C^i)$ will compute $(x \equiv y)_i$, $(y > x)_i$, $(y < x)_i$. For the case of mapping a 4-bit comparator to a set of defective crossbars, our approach yields the design in Fig. 5.6 whose design matrices P^i are shown below for convenience.

$$P^4 = \begin{pmatrix} + & 1 & x_4 & \neg x_4 & - & 1 \\ 0 & + & - & y_4 & \neg x_4 & 0 \\ 1 & \neg y_4 & 0 & 1 & 0 & - \\ \neg x_4 & 1 & - & 0 & - & x_4 \\ x_i & 0 & \neg y_4 & x_4 & 0 & 0 \\ \neg x_4 & \neg x_4 & + & \neg y_4 & y_4 & 0 \end{pmatrix} \quad P^3 = \begin{pmatrix} \neg y_3 & 1 & y_3 & + & 0 & - \\ 0 & y_3 & x_3 & y_3 & \neg x_3 & \neg y_3 \\ - & 0 & + & 0 & 0 & 0 \\ x_3 & \neg y_3 & 1 & y_3 & y_3 & \neg y_3 \\ \neg x_3 & 0 & 0 & y_3 & - & 0 \\ x_3 & + & 1 & 1 & 0 & \neg y_3 \end{pmatrix}$$

$$P^2 = \begin{pmatrix} 1 & x_2 & 1 & - & 0 & \neg x_2 \\ - & y_2 & 0 & + & \neg y_2 & 0 \\ \neg y_2 & 1 & - & 0 & - & 0 \\ \neg x_2 & \neg y_2 & 0 & 0 & y_2 & x_2 \\ y_2 & + & 1 & 1 & 0 & 0 \\ \neg x_2 & 0 & \neg x_2 & \neg y_2 & 0 & x_2 \end{pmatrix} P^1 = \begin{pmatrix} \neg x_1 & x_1 & x_1 & x_1 & + & 0 \\ y_1 & - & \neg y_1 & - & 1 & 0 \\ 0 & \neg y_1 & 0 & - & 1 & 1 \\ \neg x_1 & 1 & - & 0 & x_1 & \neg x_1 \\ \neg y_1 & 0 & 0 & y_1 & 0 & 0 \\ 0 & 1 & + & \neg y_1 & y_1 & 0 \end{pmatrix} \tag{5.20}$$

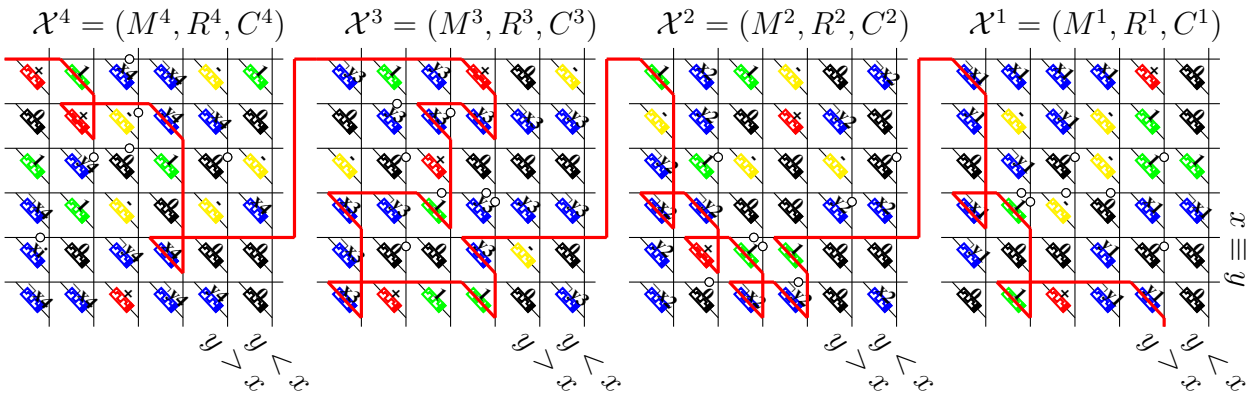


Figure 5.6: 4-bit comparator as a cascade of highly defective crossbars with mapping matrices (5.20). Black, green, red, and yellow components denote HRS, LRS, stuck-on, and stuck-off nodes, respectively. Wire breaks are denoted by white circles. The values of blue components correspond to some literal $\{x_i, \neg x_i, \quad y_i, \neg y_i\}$ as specified by the P matrix (5.20). Given two bit-vectors $x = (x_4, x_3, x_2, x_1)$ and $y = (y_4, y_3, y_2, y_1)$, we have $r_5^i \iff (x \equiv y)_i$, $c_5^i \iff (y > x)_i$, and $c_6^i \iff (y < x)_i$ and input $r_1^i = (x \equiv y)_{i+1}$, with initial input $r_1^4 = 1$. The red bars denote the flow of current under evaluation vector $\alpha = (x_4 = 1, x_3 = 1, x_2 = 0, x_1 = 0, y_4 = 1, y_3 = 1, y_2 = 0, y_1 = 1)$ when $C_5^4, C_6^4, C_5^3, C_6^3, C_5^2, C_6^2, C_5^1, C_6^1, R_5^1$ are grounded. Design generation files can be found in eecs.ucf.edu/∼velasquez/Comparator.

Recall the encouraging results of the full adder design studied in Table 5.3. If we consider the use of a network of crossbars as was done for the n-bit comparator in Fig. 5.6, it would seem obvious to exploit this same construction in order to create n-bit ripple-carry adders. However, as we demonstrate in the next chapter, this is not possible for certain operations on the homogeneous crossbars that we have looked at. Ripple-carry addition is one such operation. Fortunately, this restriction can be overcome by introducing new elements into the crossbar.

CHAPTER 6: HETEROGENEOUS CROSSBARS

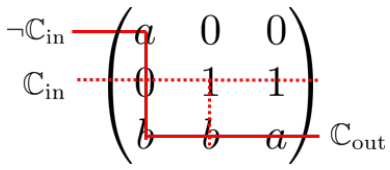
Given a crossbar $\mathcal{X} = (M, R, C)$ with source and destination wires $S, F \subseteq W$ and a formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$, we must make a distinction between the case where the input wire values are constant (i.e. $s_i = 1$ for all $\alpha \in \mathbb{B}^p$), as has been the case so far, and the case where the value of an input wire depends on the evaluation vector $\alpha \in \mathbb{B}^p$. As we established in Chapter 4, any Boolean formula can be computed on a constant-input homogeneous crossbar using paths-based logic [82]. However, as we demonstrate in Theorem 5, this universality breaks down under variable inputs. In particular, we show that ripple-carry addition cannot be computed in variable-input homogeneous crossbars. For the remainder of this chapter, we use the terms unidirectional components and diodes interchangeably.

Limitations of Paths-Based Logic

An adder is defined by its sum and carry-out bits $\mathbb{S}$ and $\mathbb{C}_{\text{out}}$. Given bits a, b, and carry-in values $\neg\mathbb{C}_{\text{in}}$ and $\mathbb{C}_{\text{in}}$, we define $(\mathbb{S}|\neg\mathbb{C}_{\text{in}}), (\mathbb{S}|\mathbb{C}_{\text{in}}), (\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}), (\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}})$ in (6.1), where $(\phi|\psi)$ is a formula denoting the value ϕ when ψ holds.

$$\begin{aligned}
\mathbb{S} =& (a \wedge \neg b \wedge \neg\mathbb{C}_{\text{in}}) \vee (\neg a \wedge b \wedge \neg\mathbb{C}_{\text{in}}) \vee \\
& (\neg a \wedge \neg b \wedge \mathbb{C}_{\text{in}}) \vee (a \wedge b \wedge \mathbb{C}_{\text{in}}) \\
\mathbb{C}_{\text{out}} =& (a \wedge b) \vee (a \wedge \mathbb{C}_{\text{in}}) \vee (b \wedge \mathbb{C}_{\text{in}}) \\
(\mathbb{S}|\neg\mathbb{C}_{\text{in}}) =& (a \wedge \neg b) \vee (\neg a \wedge b) \\
(\mathbb{S}|\mathbb{C}_{\text{in}}) =& (\neg a \wedge \neg b) \vee (a \wedge b) \\
(\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}) =& (a \wedge b), (\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}}) = a \vee b
\end{aligned} \tag{6.1}$$

The following theorem may seem esoteric, but it provides a guideline for determining which functions cannot be computed on variable-input homogeneous crossbars using paths-based logic. We motivate this limitation with a simple example. Suppose we are given a crossbar $\mathcal{X} = (M, R, C)$ with variable inputs $S = (R_1, R_2)$ and output $F = (R_3)$ such that $s_1 = \neg\mathbb{C}_{\text{in}}$, $s_2 = \mathbb{C}_{\text{in}}$, and we want $f_1 \iff \mathbb{C}_{\text{out}}$ (See equations (6.1)). Use the figure below as a reference.



There must be a path $\Pi^{S_1 \to F_1}$ to compute $(\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}) = a \wedge b$ and two paths $\Pi^{S_2 \to F_1}$, $\Pi'^{S_2 \to F_1}$ to compute the two clauses in $(\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}}) = (a) \vee (b)$. Note that when $a = 1, b = 1, \mathbb{C}_{\text{in}} = 0$, we have $s_1 = 1$ and $s_2 = 0$. Thus, path $\Pi^{S_1 \to F_1}$ will yield $f_1 = 1$ as denoted by the solid red lines in the figure. However, path $\Pi'^{S_2 \to F_1}$ will then yield $s_2 = 1$ as shown by the dotted lines, which is a contradiction since s_2 should be equal to $\mathbb{C}_{\text{in}}$. This is due to the symmetry of paths property (5.3) in homogeneous crossbars. As we will demonstrate, this limitation makes the problem of designing a ripple-carry adder on a variable-input homogeneous crossbar impossible.

Theorem 5. *There exists a class of Boolean formulas that cannot be computed on variable-input homogeneous crossbars using paths-based logic.*

Proof. Suppose there is a well-formed design P for a crossbar $\mathcal{X} = (M, R, C)$ with source and destination wires S, F for some $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and assume the following statements hold under some evaluation vector $\alpha \in \mathbb{B}^p$ for some $\beta, \omega, i, j, k, k'$: (i) $s_\beta \wedge \neg s_\omega$, (ii) ϕ^i is satisfied, (iii) $(\Phi^i_k|s_\omega) \subseteq (\Phi^i_{k'}|s_\beta)$, (iv) $(\phi^i_{k'}|s_\beta)$ is satisfied.

Since $(\Phi^i_k|s_\omega) \subseteq (\Phi^i_{k'}|s_\beta)$ and clause $(\phi^i_{k'}|s_\beta)$ is satisfied, the clause $(\phi^i_k|s_\omega)$ must also be satisfied. For every $S_h \in S$, there must exist a path $\Pi^{S_h \to F_{h'}}$ for each clause in $(\Phi^{h'}|s_h)$. Thus, there must be paths $\Pi^{S_\beta \to F_i}$, $\Pi^{S_\omega \to F_i}$ corresponding to $(\Phi^i_{k'}|s_\beta)$ and $(\Phi^i_k|s_\omega)$, respectively. Since we have chosen

α such that $(\phi^i_{k'}|s_\beta)$, $(\phi^i_k|s_\omega)$, and s_β are satisfied, implication (6.2) follows from the symmetry of paths property, yielding the contradiction $s_\omega \wedge \neg s_\omega$. $\square$

$$\left(s_\beta \wedge \bigwedge_d \pi_d^{S_\beta \to F_i} \implies f_i\right) \wedge \left(f_i \wedge \bigwedge_d \pi_d^{S_\omega \to F_i} \implies s_\omega\right) \tag{6.2}$$

An important operation that lies in this uncomputable class of functions is ripple-carry addition (6.1). Suppose we want to find a well-formed design P that computes $\mathbb{S}$, $\mathbb{C}_{\text{out}}$ for a crossbar $\mathcal{X} = (M, R, C)$ with source/destination wires S, F. Let the inputs be defined by $s_1 = \neg\mathbb{C}_{\text{in}}$ and $s_2 = \mathbb{C}_{\text{in}}$, where $(\mathbb{C}_{\text{out}}|s_1) = \{\{a, b\}\}$ and $(\mathbb{C}_{\text{out}}|s_2) = \{\{a\}, \{b\}\}$. Given evaluation vector $\alpha = (a = 1, b = 1, \mathbb{C}_{\text{in}} = 0)$, conditions (i)–(iv) from Theorem 5 hold. Indeed, we have (i) $s_1 \wedge \neg s_2$, (ii) $\mathbb{C}_{\text{out}}$ is satisfied, (iii) $(\mathbb{C}_{\text{out}_1}|s_2) \subseteq (\mathbb{C}_{\text{out}_1}|s_1)$, and (iv) $(\mathbb{C}_{\text{out}_1}|s_1)$ is satisfied.

From Theorem 5, we know that the symmetry property (5.3) yields s_2. This is clearly a contradiction since $s_1 = 1 = \neg\mathbb{C}_{\text{in}}$ and $s_2 = 0 = \mathbb{C}_{\text{in}}$ hold due to α.

Heterogeneous Crossbars

The use of heterogeneous crossbar designs defined below gives us greater control over the paths induced by allowing the use of unidirectional components. The addition of these components makes each interconnect in the crossbar a member of a trinary set and changes the dynamics of the crossbar. The flow behavior in heterogeneous crossbars follows Axiom 3. The mapping matrix P, where $P_{ij} \in \{B_1, \neg B_1, B_2, \neg B_2, \ldots, B_p, \neg B_p\} \cup \{0, 1, D\}$, satisfies both (5.4) and the additional proposition $(m_{ij} = D) \iff (P_{ij} = D)$.

Definition 6. *A heterogeneous crossbar is a crossbar $\mathcal{X} = (M = (M_{ij}), R, C)$, where each m_{ij} is chosen from a set of multiple components. For the purposes of this chapter, this would*

be the set $\{0, 1, D\}$ *of bidirectional HRS and LRS components and row-to-column unidirectional components, respectively.*

Axiom 3 (Heterogeneous Flow)**.** *Given a heterogeneous crossbar* $\mathcal{X} = (M \in \{0, 1, D\}^{|R|\times|C|}, R, C)$*,* $(r_i \wedge m_{ij} \in \{1, D\}) \implies c_j$ *and* $(c_j \wedge m_{ij} = 1) \implies r_i$ *hold. Consequently, equation (6.3) holds.*

$$\begin{aligned}&\left(\bigwedge_{j=1}^{|C|} c_j \iff \left(\bigvee_{i=1}^{|R|} (m_{ij} \in \{1, D\} \wedge r_i)\right)\right) \wedge \\ &\left(\bigwedge_{i=1}^{|R|} r_i \iff \left(\bigvee_{j=1}^{|C|} (m_{ij} = 1 \wedge c_j)\right)\right)\end{aligned} \tag{6.3}$$

It follows from (6.3) that symmetry (5.3) does not hold for a path $\Pi^{W_i \to W_j}$ with $D \in \pi^{W_i \to W_j}$. This is intuitive since D is a unidirectional component that allows the flow of current to traverse from rows to columns, but suppresses current in the opposite direction. That is, if $m_{ij} = D$, we have $(c_j \wedge m_{ij}) \not\implies r_i$. This would violate the symmetry equation (5.3). Recall that symmetry causes the contradiction in Theorem 5. In the next section, we demonstrate how this simple modification allows us to design ripple-carry adders using variable-input heterogeneous crossbars and paths-based logic.

Theorem 6. *There exists a well-formed variable-input design* P *for any* $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ *when using heterogeneous crossbars.*

Proof. We will demonstrate how to construct a well-formed design P (i.e. one where (5.5) holds). First, we construct well-formed designs k_j^i for $(\phi_j^k | s_i)$, i.e. the j^{th} clause of formula ϕ^k given input s_i. These designs are used to create the well-formed design P_i which computes $(\phi^k|s_1), \ldots, (\phi^k|s_{|S|})$, i.e. every formula ϕ^k under input s_i. These P_i designs are then used to construct the final design

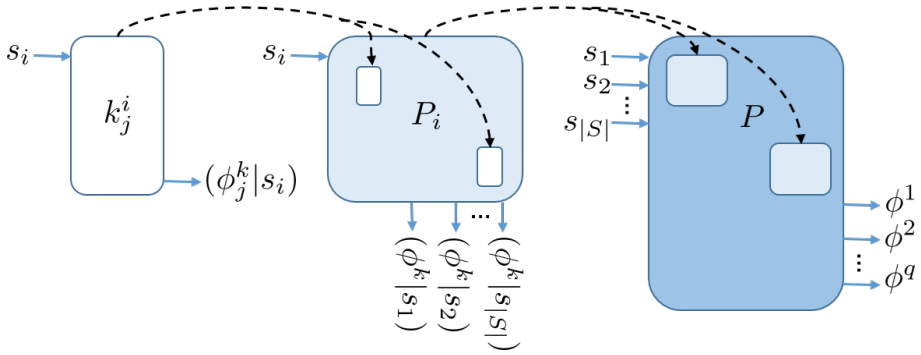


Figure 6.1: Modular representation of the construction in Theorem 6. The well-formed design k_j^i computes $(\phi_j^k|s_i)$ for a given i and is used in P_i to compute $(\phi^k|s_i)$ for all i. P_i is then used in the final design P to compute $\phi^1, \ldots, \phi^q$.

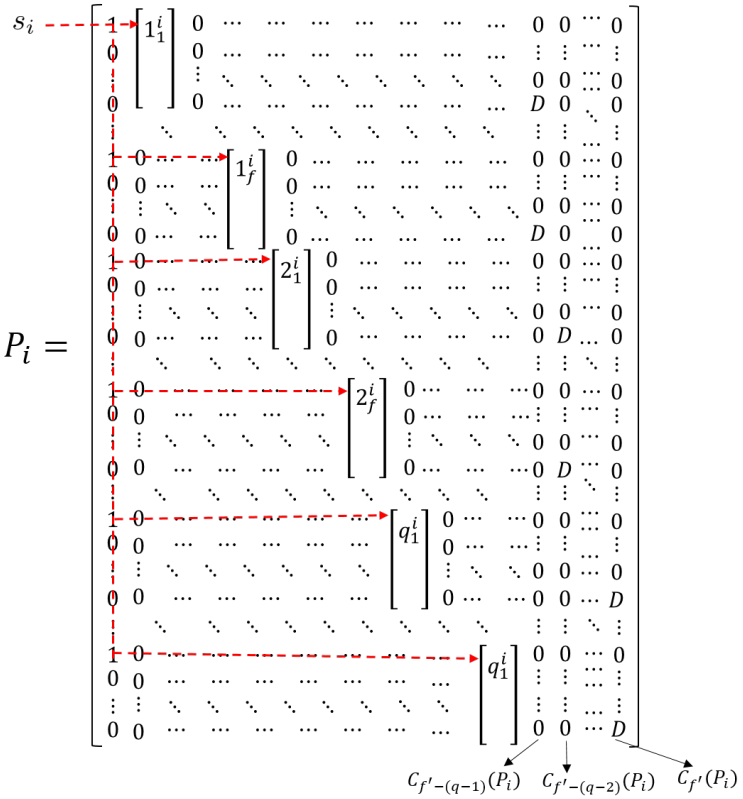
P. See Figure 6.1 for a visual representation. For the remainder of this proof, let $R_f(k_j^i)$, $R_f(P_i)$, $R_f(P)$ and $C_{f'}(k_j^i)$, $C_{f'}(P_i)$, $C_{f'}(P)$ denote the last rows and columns for the designs mentioned.

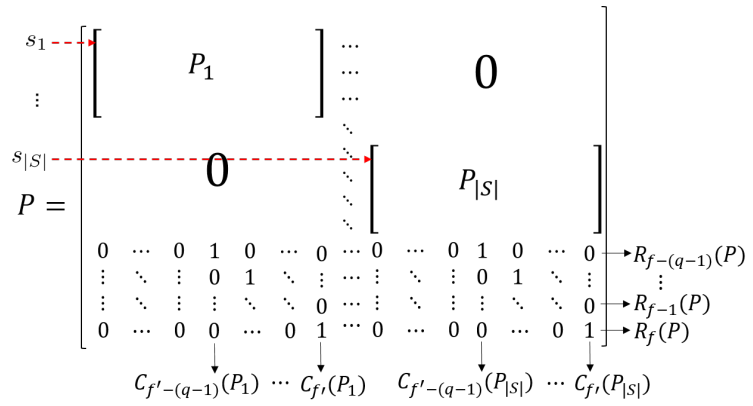
Let k_j^i denote a well-formed design for a crossbar with source wire $R_1(k_j^i)$ and destination wire $R_f(k_j^i)$ that computes $(\phi_j^k|s_i)$. Given $r_1(k_j^i) = s_i$, there exists a path such that $r_1(k_j^i) \wedge \bigwedge_d \pi^{R_1(k_j^i) \to R_f(k_j^i)} \implies r_f(k_j^i)$ and $r_f(k_j^i) \iff (\phi_j^k|s_i)$ for all evaluations $\alpha \in \mathbb{B}^p$. Since $(\phi_j^k|s_i)$ is simply a conjunction of variables, the mapping shown below suffices to compute it.

$$k_j^i = \begin{array}{c} \\ R_1(k_j^i) \\ R_2(k_j^i) \\ R_3(k_j^i) \\ \vdots \\ R_f(k_j^i) \\ \\ C_1(k_j^i) \end{array} \begin{array}{c} \\ \begin{pmatrix} \phi_{j1}^k & 0 & 0 & 0 & \ldots & 0 \\ \phi_{j2}^k & \phi_{j3}^k & 0 & 0 & \ldots & 0 \\ 0 & \phi_{j4}^k & \phi_{j5}^k & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & \phi_{j*}^k \end{pmatrix} \\ \\ C_2(k_j^i) \quad C_3(k_j^i) \quad C_4(k_j^i) \quad \ldots \quad C_{f'}(k_j^i) \end{array}$$

the design P_i computes formulas $(\phi^1|s_i), \ldots, (\phi^q|s_i)$ for a crossbar with source wire $R_1(P_i)$ and destination wire set $F(P_i) = (C_{f'-(q-1)}(P_i), \ldots, C_{f'}(P_i))$. Note that component M_{11} is in the

LRS state $m_{11} = 1$. Thus, it follows from Axiom 3 that $c_1(P_i) = 1$ if $s_i = 1$ and, consequently, every $r_1(k_j^i)$ will be 1 if $c_1(P_i) = 1$. This is due to all of the LRS components in the first column of P_i. If $(\phi_j^k | s_i)$ evaluates to 1, then the design k_j^i will yield $r_f(k_j^i) = 1$, which in turn will yield $c_{f'-(k-1)}(P_i) = 1$ due to the unidirectional components on the rightmost columns of P_i.





Design P computes $\phi^1, \ldots, \phi^q$ for a crossbar with source and destination wire sets $S = (R_1(P_1), \ldots,$

$R_1(P_{|S|})$) and $F = (R_{f-(q-1)}(P), \ldots, R_f(P))$, respectively. Note that there are paths induced by P_i and P such that (6.4) holds.

$$\begin{aligned}
&(\phi^q | s_i) \implies c_{f'}(P_i) \implies r_f(P) \wedge \\
&(\phi^{q-1} | s_i) \implies c_{f'-1}(P_i) \implies r_{f-1}(P) \wedge \\
&\qquad\qquad \vdots \\
&(\phi^2 | s_i) \implies c_{f'-(q-2)}(P_i) \implies r_{f-(q-2)}(P) \wedge \\
&(\phi^1 | s_i) \implies c_{f'-(q-1)}(P_i) \implies r_{f-(q-1)}(P)
\end{aligned} \tag{6.4}$$

This establishes condition (5.5) in one direction. That is, we have $\bigwedge_k r_{f-(q-k)}(P) \impliedby \phi^k$. In order to finalize the proof, we must show that $\bigwedge_k r_{f-(q-k)}(P) \implies \phi^k$. Note that $r_{f-(q-k)}$ holds iff there is some i for which $c_{f'-(q-k)}(P_i)$ is true. In turn, $c_{f'-(q-k)}(P_i)$ only holds if there are k, j such that $r_f(k^i_j)$ is true. We know that $r_f(k^i_j) \iff \phi^k_j \implies \phi^k$. Therefore, we conclude that $r_f(P) \iff \phi^q$, $r_{f-1}(P) \iff \phi^{q-1}$, $\ldots$, $r_{f-(q-1)}(P) \iff \phi^1$ as intended. See Fig. 6.2 for an example. □

Fig. 6.2 helps to elucidate the importance of unidirectional components. Indeed, suppose that every $m_{ij} = D$ in the design presented in Fig. 6.2 is replaced with $m_{ij} = 1$ and let $\alpha = (a = 1, b = 1, \mathbb{C}_{\text{in}} = 0)$. Note that there would be a path $\Pi' = (M_{16,13}, M_{14,13}, M_{14,11}, M_{13,11}, M_{13,7}, M_{7,7})$ such that $r_1 \wedge \bigwedge_d \pi_d^{R_1 \to R_{16}} \wedge \bigwedge_d \pi'_d \implies r_7$, contradicting $r_7 = \mathbb{C}_{\text{in}}$.

Design Automation

Theorem 6 establishes the universality of variable-input crossbars using paths-based logic when heterogeneous components are used. However, the design methodology proposed therein leads to

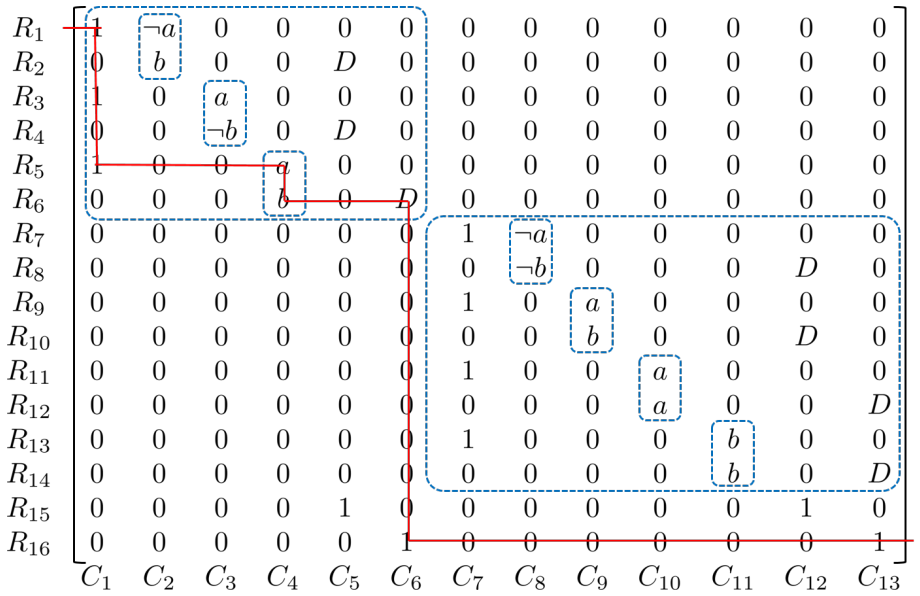


Figure 6.2: Variable-input heterogeneous design for a full adder, where $\mathbb{S}$, $\mathbb{C}_{\text{in}}$, and $\mathbb{C}_{\text{out}}$ denote the sum, carry-in, and carry-out bits, and a, b are the bits to be added. This design follows from the construction given in Theorem 6. We have $r_1 = \neg\mathbb{C}_{\text{in}}$, $r_7 = \mathbb{C}_{\text{in}}$, $r_{15} \iff \mathbb{S}$, and $r_{16} \iff \mathbb{C}_{\text{out}}$ for all evaluations $\alpha \in \mathbb{B}^3$ (see equations (6.1)). Blue rectangles denote constructions k_j^i and P_i from the proof and the red trajectory delineates the path $\Pi^{R_1 \to R_{16}}$ that makes r_{16} true under evaluation vector $\alpha = (a = 1, b = 1, \mathbb{C}_{\text{in}} = 0)$.

spatially complex crossbars as is evidenced by Fig. 6.2. This is not surprising since we encountered the same issue when a universal methodology was proposed for constant-input homogeneous crossbars. This motivates the need for a design automation approach to produce compact crossbar designs given a set of formulas to compute. We accomplish this in a similar fashion to the previous chapter. That is, by specifying the dynamics of variable-input heterogeneous crossbars using temporal logic and framing the design synthesis problem as a bounded model checking (BMC) problem.

The initial state I (6.5) and transition relation τ (6.6) are modified to account for unidirectional

components. The specification ψ (5.10) and BMC formula $\mathcal{M}_{\text{BMC}}$ (5.7) remain unchanged.

$$\begin{aligned}(\mathbb{S}_i|\neg\mathbb{C}_{i-1}) =&(x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i)\\ (\mathbb{S}_i|\mathbb{C}_{i-1}) =&(\neg x_i \wedge \neg y_i) \vee (x_i \wedge y_i)\\ (\mathbb{C}_i|\neg\mathbb{C}_{i-1}) =&(x_i \wedge y_i), (\mathbb{C}_i|\mathbb{C}_{i-1}) = x_i \vee y_i\end{aligned} \tag{6.7}$$

By providing the model checking formula $\mathcal{M}_{\text{BMC}}$ (5.7) to the NuSMV 2.6.0 model checker [83], we synthesized a mapping for an n-bit ripple-carry adder based on equations (6.7) given two n-bit vectors $x, y \in \{0,1\}^n$. The well-formed design P^i for the crossbar $\mathcal{X}^i = (M^i, R^i, C^i)$ with source and destination wires $S = (R^i_1, R^i_2), F = (R^i_4, R^i_5, C^i_5)$ computes $(\neg\mathbb{C}_i, \mathbb{C}_i, \mathbb{S}_i)$, where the inputs are defined as $s^i_1 = \neg\mathbb{C}_{i-1}$ and $s^i_2 = \mathbb{C}_{i-1}$. This design is captured by equation (6.8). See Fig. 6.3 for an example of how this design can be used in a 4-bit adder.

$$\begin{aligned}I(u_0) \triangleq \bigwedge_{\alpha\in\mathbb{B}^p}\left(\bigwedge_{w_i\in S}(\mathcal{L}^0_\alpha[w_i] \iff s_i)\right) \wedge \bigwedge_{w_i\notin S}\neg\mathcal{L}^0_\alpha[w_i] \wedge \left(\bigwedge_{i,j}\mathcal{L}_\alpha[m_{ij}] = D \iff P_{ij} = D\right) \wedge \\ \left(\bigwedge_{i,j,k}(\mathcal{L}_\alpha[m_{ij}] = 1) \iff (P_{ij} = 1) \vee ((P_{ij} = B_k) \wedge \mathcal{L}_\alpha[b_k]) \vee ((P_{ij} = \neg B_k) \wedge \neg\mathcal{L}_\alpha[b_k])\right)\end{aligned} \tag{6.5}$$

$$\begin{aligned}\tau(u_t, u_{t+1}) \triangleq \bigwedge_{\alpha\in\mathbb{B}^p}\left(\bigwedge_i\left(\mathcal{L}^{t+1}_\alpha[r_i] \iff \bigvee_j \mathcal{L}_\alpha[m_{ij}] = 1 \wedge \mathcal{L}^t_\alpha[c_j]\right) \wedge \right. \\ \left.\bigwedge_j\left(\mathcal{L}^{t+1}_\alpha[c_j] \iff \bigvee_i(\mathcal{L}_\alpha[m_{ij}] \in \{1, D\}) \wedge \mathcal{L}^t_\alpha[r_i]\right)\right)\end{aligned} \tag{6.6}$$

$$\begin{pmatrix} D & 0 & 0 & 0 & 0 \\ 0 & y_i & y_i & \neg y_i & 0 \\ D & 0 & x_i & \neg x_i & 1 \\ 1 & 0 & \neg y_i & y_i & 0 \\ \neg x_i & 0 & \neg y_i & 0 & 0 \\ 0 & \neg x_i & 0 & x_i & 0 \end{pmatrix} \tag{6.8}$$

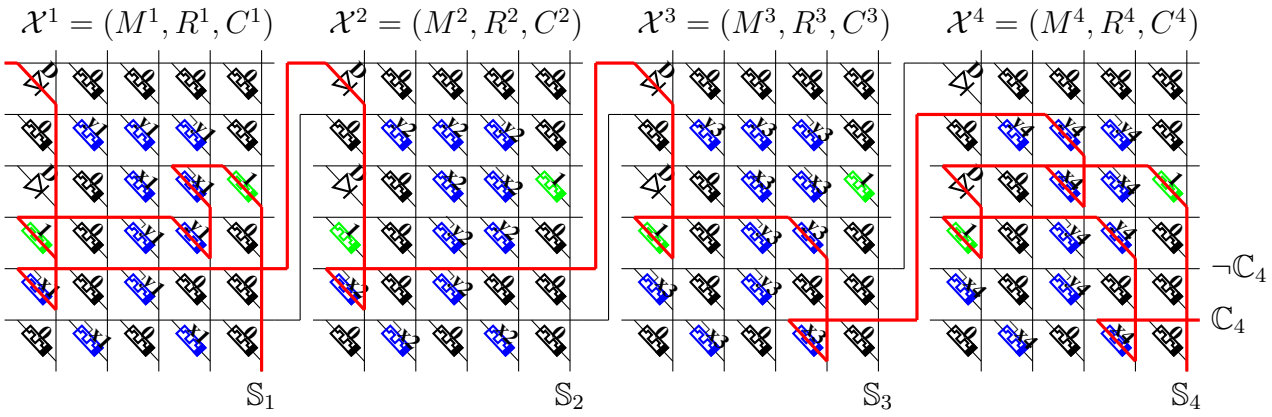


Figure 6.3: 4-bit ripple-carry adder as a cascade of crossbars with mapping matrix (6.8). Given two bit-vectors $X = (x_4, x_3, x_2, x_1)$ and $Y = (y_4, y_3, y_2, y_1)$, we have $r_5^i \iff \neg\mathbb{C}_i$, $r_6^i \iff \mathbb{C}_i$, and $c_5^i \iff \mathbb{S}_i$ and inputs $r_1^i = \neg\mathbb{C}_{i-1}$ and $r_2^i = \mathbb{C}_{i-1}$ with $r_1^1 = \neg\mathbb{C}_0 = 1$ since there is no carry-in bit for the least significant bit addition. The red bars denote the flow of current under evaluation vector $\alpha = (x_4 = 1, x_3 = 1, x_2 = 0, x_1 = 0, y_4 = 1, y_3 = 1, y_2 = 0, y_1 = 1)$ when wires $C_5^1, C_5^2, C_5^3, C_5^4, R_5^4, R_6^4$ are grounded. Note that the values read on these wires are $\mathbb{S}_1 = 1, \mathbb{S}_2 = 0, \mathbb{S}_3 = 0, \mathbb{S}_4 = 1, \neg\mathbb{C}_4 = 0, \mathbb{C}_4 = 1$, respectively, yielding the correct result. That is, given $x = 12$ and $y = 13$, we read the value 25. After applying a voltage pulse of $5V$ to R_1^1, we obtain the following voltage values with respect to ground for the grounded wires: $4.5751V, 39.9629mV, 34.0887mV, 3.7873V, 9.7456mV$, $3.6443V$. Design and circuit generation files can be found in eecs.ucf.edu/ ∼velasquez/HeteroAdder.

In [84], it was shown that a crossbar $\mathcal{X} = (M, R, C)$ can be configured in $\min\{|R|, |C|\} + 1$ steps. Thus, each crossbar in our n-bit adder design can be programmed independently in 6 steps and a read voltage is then applied to read the values of the output wires in F. Therefore, we can

compute n-bit addition using a constant number of steps. See Table 6.1 for a comparison with other approaches.

Table 6.1: Comparison of our crossbar ripple-carry adder (XRCA) against other crossbar n-bit adder designs proposed in the literature. Our design is state-of-the-art in terms of execution steps required to compute n-bit addition.

	[74]	[71]	[69]	[69]	[67]	[68]	[72]	[72]	[73]	[73]	XRCA
Execution Steps	$19n$	$8n+12$	$29n$	$5n+18$	$89n$	$15n$	$13n+2$	$7n+21$	$2n+4$	$4n+5$	7
Crossbar Nodes	$5n+1$	$35n$	$3n+3$	$9n$	$3n+5$	$5n+6$	$3n+6$	$8n$	$2n+2$	$n+2$	$30n$

Fault Tolerance

The same stuck-on and stuck-off defect specifications that were introduced for homogeneous crossbars apply in the heterogeneous domain. However, we should now account for diode placement in our specifications. This is important for the sake of efficient crossbar reconfiguration. Indeed, suppose that ϕ is mapped to a crossbar at some point. However, computational demands change and now ϕ' must be mapped to this same crossbar. An ideal mapping would utilize the diodes already placed on the crossbar; otherwise, the fabrication costs associated with removing and/or adding components could become prohibitively expensive. To this end, we extend specification (5.14) to the following (6.9).

$$\bigwedge_{i=1}^{|R|}\bigwedge_{j=1}^{|C|}(z_{ij}=+ \implies p_{ij}=1)\wedge(z_{ij}=- \implies p_{ij}=0)\wedge(z_{ij}=\downarrow \implies p_{ij}=D) \tag{6.9}$$

Broken wires are also accounted for differently in the case of heterogeneous crossbars due to the

presence of unidirectional components. As such, specification (5.16) is changed to (6.10).

$$\begin{aligned}\bigwedge_{\alpha\in\mathbb{B}^p}\bigwedge_{i=1}^{|R|}\left(\mathcal{L}_\alpha^{t+1}[r_{i,m_{ij}^c}] \iff \bigvee_{j=1}^{|C|}\left(\mathcal{L}_\alpha[m_{ij}]=1\wedge\mathcal{L}_\alpha^t[c_{j,m_{ij}^r}]\right)\right)\wedge\\ \bigwedge_{j=1}^{|C|}\left(\mathcal{L}_\alpha^{t+1}[c_{j,m_{ij}^r}] \iff \bigvee_{i=1}^{|R|}\left(\mathcal{L}_\alpha[m_{ij}]\in\{1,D\}\wedge\mathcal{L}_\alpha^t[r_{i,m_{ij}^c}]\right)\right)\end{aligned} \tag{6.10}$$

As an example, suppose we are given a highly defective 8×8 crossbar $\mathcal{X}=(M,R,C)$ with source wires $S=(R_1,R_2)$ and destination wires (R_7,R_8,C_8) such that $s_1=\neg\mathbb{C}_{i-1}$, $s_2=\mathbb{C}_{i-1}$, and we want to design a full adder as defined by equations (6.7). Let the wire breaks be defined by $\theta^{R_2}=\theta^{C_3}=(1,2,9),\theta^{R_3}=(1,4,9),\theta^{R_5}=(1,7,9)$ and $\theta^{C_2}=(1,3,6,9),\theta^{C_6}=(1,5,9)$. Let the stuck faults be given by $z_{17}=z_{35}=z_{66}=z_{86}=+$, $z_{21}=z_{27}=z_{38}=z_{51}=z_{63}=z_{64}=z_{71}=z_{85}=-$, and $z_{45}=\downarrow$. Using the bounded model checking approach presented, we have generated the design (6.11). See Fig. 6.4 for a visualization.

$$\begin{pmatrix} 0 & y_i & \neg x_i & y_i & y_i & x_i & + & 0\\ - & \neg y_i & 0 & \neg y_i & 0 & y_i & - & \neg x_i\\ \neg x_i & \neg y_i & 1 & 0 & + & y_i & 0 & -\\ x_i & 0 & \neg y_i & \neg y_i & \downarrow & \neg x_i & 0 & y_i\\ - & y_i & \neg y_i & 1 & 0 & 0 & \neg y_i & x_i\\ 0 & 0 & - & - & 0 & + & 0 & 0\\ - & \neg y_i & 0 & 0 & \neg x_i & 0 & \neg y_i & 0\\ 0 & y_i & y_i & x_i & - & + & 0 & 0 \end{pmatrix} \tag{6.11}$$

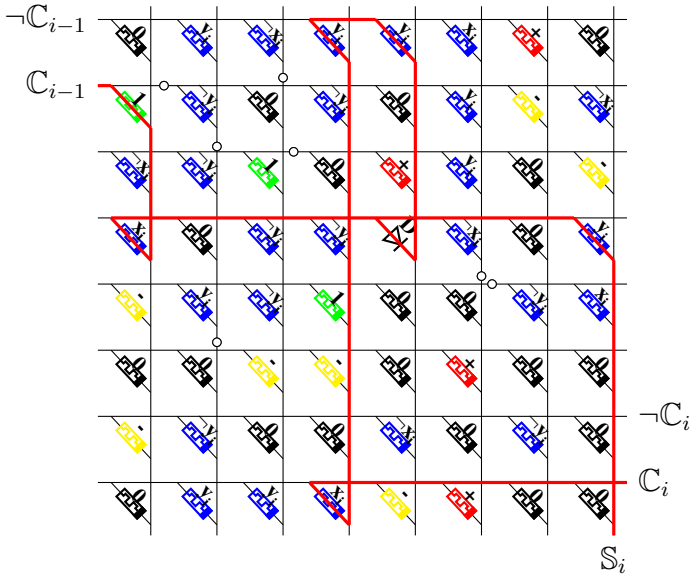


Figure 6.4: Variable-input crossbar design P for a crossbar $\mathcal{X} = (M, R, C)$ with input/output wires $S = (R_1, R_2), F = (R_7, R_8, C_8)$ for a full adder $\phi = (\neg\mathbb{C}_i, \mathbb{C}_i, \mathbb{S}_i)$ (6.11) given inputs $r_1 = \neg\mathbb{C}_{i-1}$ and $r_2 = \mathbb{C}_{i-1}$ corresponding to the carry bits of the previous addition operation. Black, green, red, and yellow components denote HRS, LRS, stuck-on, and stuck-off nodes, respectively. The values of blue components correspond to some variable $\{x_i, \neg x_i, y_i, \neg y_i\}$ as specified by the P matrix (6.11). Wire breaks are represented by $\theta^{R_2} = \theta^{C_3} = (1, 2, 9), \theta^{R_3} = (1, 4, 9), \theta^{R_5} = (1, 7, 9)$ and $\theta^{C_2} = (1, 3, 6, 9), \theta^{C_6} = (1, 5, 9)$. These breaks are denoted by white circles. The red bars denote the paths under evaluation vector $\alpha = (x_i = 1, y_i = 1, \mathbb{C}_{i-1} = 1)$.

Experimental Results

We utilize HSPICE for our experiments. For each $m_{ij} = 1$ ($m_{ij} = 0$), we use resistors with LRS (HRS) resistance $R_{LRS} = 10\Omega$ ($R_{HRS} = 1M\Omega$) and we implement the SDM02U30CSP diode model from Diodes Incorporated® [85] for unidirectional components $m_{ij} = D$. Resistors-to-ground with resistance $R_G = 500\Omega$ are placed before each grounded wire. See Table 6.2 and Fig. 6.3 for simulation results of different adder circuits.

We use the NVSim non-volatile memory simulator [7] to test the energy efficiency of our approach against conventional CMOS adders (See Table 6.3). For our interconnects, we use the memristor

Table 6.2: HSPICE simulation results for the full adder design in (6.8). Each column entry denotes values under an evaluation vector $\alpha = (x_i, y_i, \mathbb{C}_{i-1})$. Wires C_5, R_5, R_6 are grounded and a $5V$ voltage pulse is applied to R_1 if $\mathbb{C}_{i-1} = 0$ or to R_2 if $\mathbb{C}_{i-1} = 1$. The voltage values read correspond to $\mathbb{S}_i, \neg\mathbb{C}_i, \mathbb{C}_i$, respectively. Each entry denotes the voltage reading obtained from grounded wires C_5, R_5, R_6.

	000	001	010	011	100	101	110	111
C_5	$17.4m$	4.544	4.627	$12.4m$	4.717	$14.7m$	$22.1m$	4.551
R_5	4.902	4.567	4.718	$12.5m$	4.807	$14.7m$	$4.97m$	$11.7m$
R_6	$9.89m$	$18.6m$	$24.4m$	4.807	$28.9m$	4.807	4.807	4.38

studied in [86], which has switching time and energy of 85 ps and 3 fJ, respectively, under a 2 V voltage pulse. The results can be seen in Table 6.3. Note that the power-delay product of our adder is less than half that of the next most efficient CMOS adder, and this does not take into account the energy consumed to fetch the operands from memory.

Concluding Remarks

We have presented a design automation framework for generating logic-in-memory heterogeneous crossbar designs using bounded model checking. The necessity for heterogeneity was motivated by defining a class of functions that cannot be computed using variable-input homogeneous crossbars and paths-based logic. We demonstrated the effectiveness of the proposed approach by generating ripple-carry adder circuits that are state-of-the-art in terms of execution steps. We also showed how our approach is easily extended to handle various common defects that arise in self-assembled crossbars.

Table 6.3: Power comparison between the proposed crossbar ripple-carry adder architecture (XRCA) and adders presented in the survey [4] using traditional CMOS. These are ripple-carry (RCA), increment (INCA), triangle (TRIA), uniform and progressive carry-select (CSELA-UNIF, CSELA-PROG), conditional (COND), uniform and progressive-carry bypass (CBYPASS-UNIF, CBYPASS-PROG), and ripple-carry and hierarchical-carry lookahead (CLA-RIPPLE, CLA-HIER) adders. All architectures are simulated using 180 nm technology with a 1.8 V pulse and 10 MHz frequency. NVSIM files can be found in eecs.ucf.edu/$\sim$velasquez/Table6.3.

16-Bit Adder	Delay (ns)	Power (μW)	PDP ($\times 10^{-15}$)
RCA	8.764	5.134	44.99
INCA	7.413	6.369	47.21
TRIA	9.565	21.07	201.5
CSELA-UNIF	5.198	11.64	60.50
CSELA-PROG	5.095	12.74	64.91
COND	4.755	24.93	118.5
CBYPASS-UNIF	5.818	7.417	43.15
CBYPASS-PROG	4.432	9.020	39.98
CLA-RIPPLE	8.189	8.630	70.67
CLA-HIER	5.498	11.66	64.11
XRCA	0.777	24.42	18.97

CHAPTER 7: DESIGN AUTOMATION OF VOLTAGE SEQUENCES

In this chapter, we utilize the same bounded model checking technique that has been presented thus far. However, we now search the space of voltage sequences that can be applied to the memory so that a desired result will be stored within it. The underlying memory architecture differs from previous chapters in that we will now use a linear array of memory cells. This can be thought of as a crossbar with a single row wire and many column wires.

While the approach presented herein applies to Boolean formulas in general, we present results for addition circuits. This allows us to compare our designs with several methods in the literature. Indeed, we answer the following question. Given an initial memory configuration of predefined size containing two bit-vectors $\mathbf{x}$ and $\mathbf{y}$, what is the shortest sequence of voltage pulses that will store the sum and carry bits of $\mathbf{x} + \mathbf{y}$ in the memory? Note that the answer to such a question is agnostic to the preservation or destruction of the values in the initial memory configuration. Indeed, we consider the problem of designing n-bit adders under varying degrees of destructive operation. The non-destructive (ND) design preserves stored data by storing the sum in a reserved section of memory. The semi-destructive (SD) design reduces the number of computational components by overwriting the memory storing one of the inputs and retaining the data for the other input. The fully-destructive (FD) design does not attempt to retain either input, which reduces the number of execution steps needed.

Designs for each adder are synthesized by posing the ND, SD, and FD problems as bounded model checking procedures. By defining the initial memory configuration, a valid set of voltage inputs, and modeling the memristor switching dynamics as logical formulas, the model checker searches the space of valid sequences of voltage inputs to the wires in the memory such that the addition results are stored. In particular, it returns the shortest such sequence. Thus, our approach generates

Adder	Number of Nodes	Execution Steps
[74]	$5n + 1$	$19n$
[71]	$35n$	$8n + 12$
[69]	$3n + 3$	$29n$
[67]	$3n + 5$	$89n$
[68]	$5n + 6$	$15n$
Non-destructive	$3n + 3$	$7n$
Semi-destructive	$2n + 3$	$7n$
Fully destructive	$2n + 3$	$6n$

Table 7.1: Comparison of our adder designs against other n-bit in-memory adder designs proposed in the literature.

an addition procedure that is optimal in the number of execution steps given the number of nodes in the memory.

Traditionally, logic synthesis algorithms have been used to design large memristive circuits by reducing the problem to a combination of atomic logic gates, such as AND-INVERTER [87] or MAJORITY-INVERTER [88] graphs. Rules of Boolean optimization are then applied to reduce the size and delay of the resulting design. It is not clear how such an approach could leverage the capability of performing destructive operations in applications that permit data loss. Our approach to the design of memristive circuits does not reduce and transform a large logical Boolean formula in terms of template designs implementing a specific fundamental logical operation. Instead, given a memory array of predefined size, we use formal methods to discover the shortest sequence of input operations that directly implement a given logical formula. In doing so, we trade off the generality of logic synthesis methods for greater parsimony in space and time as well as for the capability of exploiting the use of destructive operations.

Methodology

We first define the topology of our memory as a linear array of memristors as shown in Figure 7.2 at the end of this chapter. In this model, a common wire u is connected to a load resistor R_G, and each memristor m has one terminal connected to u. The other terminal of m is connected to a tri-state voltage driver that serves as input to the memristor. This input voltage is encoded by the wire $w \in \{V_{LOW}, V_{HI}, Z\}$, where $V_{OFF} < V_{LOW} < V_{ON} < V_{HI}$ and Z denotes high-impedance. Additionally, we define V_{HI} and V_{LOW} such that $V_{HI} - V_{LOW} > V_{ON}$ and $V_{LOW} - V_{HI} < V_{OFF}$. We encode the state of the common wire as a Boolean value such that $u = 1$ if it has a voltage of significant magnitude and $u = 0$ otherwise. Similarly, the resistance states of m are encoded by $m = 0$ for R_{OFF} and $m = 1$ for R_{ON}.

Now, let $\mathbf{x} = x_1 x_2 \dots x_n$ and $\mathbf{y} = y_1 y_2 \dots y_n$ be the two n-bit input vectors to the adder, and let c denote the carry-bit. Furthermore, let a_1 and a_2 denote auxiliary work units used to help calculate the sum $\mathbf{x} + \mathbf{y}$. This choice is not arbitrary as it has been shown that two auxiliary memristors are sufficient to compute any Boolean formula [89]. Let $\mathbf{s} = s_1 s_2 \dots s_n$ denote the desired sum to be computed. Finally, let m_i^{α} denote the state of the memristor used to encode the i^{th} element of some variable α, and let w_i^{α} denote the state of the corresponding input wire to the memristor. Then, we can define an arbitary input and memory state of our system model by the respective k-tuples W^{type} and M^{type} as follows, where $type \in \{\text{ND}, \text{SD}, \text{FD}\}$ denotes the nature of the circuit being synthesized – non-destructive, semi-destructive or fully-destructive:

$$W^{\text{ND}} = (w_n^x, \dots, w_1^x, w_n^y, \dots, w_1^y, w^c, w_1^a, w_2^a, w_n^s, \dots, w_1^s)$$

$$M^{\text{ND}} = (m_n^x, \dots, m_1^x, m_n^y, \dots, m_1^y, m^c, m_1^a, m_2^a, m_n^s, \dots, m_1^s)$$

$$W^{\text{SD}} = W^{\text{FD}} = (w_n^x, \dots, w_1^x, w_n^y, \dots, w_1^y, w^c, w_1^a, w_2^a)$$

$$M^{\text{SD}} = M^{\text{FD}} = (m_n^x, \dots, m_1^x, m_n^y, \dots, m_1^y, m^c, m_1^a, m_2^a).$$

We define the initial state of the system with no voltage on the common wire, i.e. $u = 0$, and the memory components configured in accordance with M_{init}^{type}. We then seek to learn a sequence of input values W^{type} such that the memory configuration defined by M_{final}^{type} is eventually obtained. The states M_{init}^{type} and M_{final}^{type} are defined as follows, where the $*$ symbol denotes a *don't-care* value.

$$M_{\text{init}}^{\text{ND}} = (x_n, \ldots, x_1, y_n, \ldots, y_1, 0, 0, 0, 0, \ldots, 0)$$
$$M_{\text{final}}^{\text{ND}} = (x_n, \ldots, x_1, y_n, \ldots, y_1, c_{\text{out}}, *, *, s_n, \ldots, s_1)$$
$$M_{\text{init}}^{\text{SD}} = M_{\text{init}}^{\text{FD}} = (x_n, \ldots, x_1, y_n, \ldots, y_1, 0, 0, 0)$$
$$M_{\text{final}}^{\text{SD}} = (x_n, \ldots, x_1, s_n, \ldots, s_1, c_{\text{out}}, *, *)$$
$$M_{\text{final}}^{\text{FD}} = (*, \ldots, *, s_n, \ldots, s_1, c_{\text{out}}, *, *)$$

Given the state of the system at some time t, we define a transition relation that describes the state behavior of the common wire u and captures the switching dynamics of a component memristor m from t to $t + 1$.

The voltage (with respect to ground) on the common wire u will be negligible (i.e. $u = 0$) unless there exists some ON memristor $m = 1$ with the corresponding input $w \neq Z$. In the case that $w = V_{LOW}$, the resulting voltage on the common wire due to this memristor is not significant, so again we have $u = 0$. Therefore, we obtain $u = 1$ if and only if some input $w = V_{HI}$ and its corresponding memristor is ON (i.e. $m = 1$). For simplicity, the voltage across u can be defined by $V_u \approx V_{HI}$ when $u = 1$ and $V_u \approx V_{LOW}$ when $u = 0$.

The dynamic switching of a memristor m is dependent on the electric potential difference across its terminals and its current state; so, we can define this characteristic behavior with respect to the states of w and the common wire u as shown in Figure 7.1. In particular, $w = Z$ acts as an identity operator by preventing any significant voltage across m. Hence, any component memristor m in an

w_t	m_t	u_t	m_{t+1}
Z	R_s	$*$	R_s
V_{LOW}	0	$*$	0
V_{LOW}	1	0	1
V_{LOW}	1	1	0
V_{HI}	0	0	1
V_{HI}	0	1	0
V_{HI}	1	$*$	1

Figure 7.1: State transitions for a memristor m given the voltage on its input wire w and the state of the common wire u.

arbitrary resistance state R_s will retain that state, regardless of the state of u, if its respective input $w = Z$. As we consider the voltage on the common wire $V_u \approx V_{HI}$ when $u = 1$, a component m will retain or switch to the R_{OFF} state if $u = 1$ and its corresponding $w = V_{LOW}$ because it will be the case that $V_{LOW} - V_u < V_{OFF}$. Conversely, m will retain or switch to the R_{ON} state if $u = 0$ and $w = V_{HI}$ because $V_u \approx V_{LOW}$, so $V_{HI} - V_u > V_{ON}$. The resistance state of m is unaltered in all other cases.

Now, consider the following sub-computation of the k^{th} sum bit s_k and carry-bit c.

$$s_k = x_k \oplus y_k \oplus c_{\text{in}}$$

$$c = c_{\text{out}} = (x_k \wedge y_k) \vee (x_k \wedge c_{\text{in}}) \vee (y_k \wedge c_{\text{in}})$$

We define $\mathbb{W}^{type} \subset W^{type}$ and $\mathbb{M}^{type} \subset M^{type}$ to be the respective set of input wires and memory

components needed to perform this addition as follows.

$$\mathbb{W}^{\text{ND}} = (w_k^x, w_k^y, w^c, w_1^a, w_2^a, w_k^s)$$
$$\mathbb{M}^{\text{ND}} = (m_k^x, m_k^y, m^c, m_1^a, m_2^a, m_k^s)$$
$$\mathbb{W}^{\text{SD}} = \mathbb{W}^{\text{FD}} = (w_k^x, w_k^y, w^c, w_1^a, w_2^a)$$
$$\mathbb{M}^{\text{SD}} = \mathbb{M}^{\text{FD}} = (m_k^x, m_k^y, m^c, m_1^a, m_2^a)$$

From the state transitions in Figure 7.1, we can isolate the execution of this sub-computation by setting $w = Z$ for all $w \notin \mathbb{W}^{type}$. Furthermore, a solution to this sub-problem, which is merely the implementation of a 1-bit full adder, can be applied iteratively over the entire system to implement the full n-bit adder. Thus, we can obtain a solution to the generalized ND, SD, and FD design problems via a reduction to their respective 1-bit full adder equivalents.

To perform this reduction, we utilize the same transition relation (Fig. 7.1) but redefine our initial, and corresponding final, memory configurations by $\mathbb{M}_{\text{init}}^{type}$ and $\mathbb{M}_{\text{final}}^{type}$ as follows.

$$\mathbb{M}_{\text{init}}^{\text{ND}} = (x_k, y_k, c_{\text{in}}, 0, 0, 0)$$
$$\mathbb{M}_{\text{final}}^{\text{ND}} = (x_k, y_k, c_{\text{out}}, *, *, s_k)$$
$$\mathbb{M}_{\text{init}}^{\text{SD}} = \mathbb{M}_{\text{init}}^{\text{FD}} = (x_k, y_k, c_{\text{in}}, 0, 0)$$
$$\mathbb{M}_{\text{final}}^{\text{SD}} = (x_k, s_k, c_{\text{out}}, *, *)$$
$$\mathbb{M}_{\text{final}}^{\text{FD}} = (*, s_k, c_{\text{out}}, *, *)$$

Let $\mathbb{W}_t^{type}$ and $\mathbb{M}_t^{type}$ define the states of the input wires and memory components at time t. Let $\mathbb{M}_1^{type} = \mathbb{M}_{\text{init}}^{type}$, and let t_{max} define the time at which the addition operation is completed. Observe that for $t > 1$, $\mathbb{M}_t^{type}$ is the memory configuration resulting from the application of input voltages

$\mathbb{W}_{t-1}^{type}$. Therefore, our goal is to find a sequence of inputs $\mathbb{W}_1^{type}, \mathbb{W}_2^{type}, \ldots, \mathbb{W}_{t_{\text{max}}-1}^{type}$ that yields $\mathbb{M}_{t_{\text{max}}}^{type} = \mathbb{M}_{\text{final}}^{type}$ such that t_{max} is minimized.

Given the logical definition of our system, the adder design synthesis can be encoded as a bounded model checking (BMC) problem. In particular, the BMC algorithm encodes the system dynamics (Fig. 7.1) as a propositional formula and leverages satisfiability solvers to verify if the property $\mathbb{M}_{\text{final}}^{type}$ holds given an initial condition $\mathbb{M}_{\text{init}}^{type}$ for all possible inputs. A solution to this problem takes the form of an instantiation of the voltage sequences $\mathbb{W}^{type}$ that derive $\mathbb{M}_{\text{final}}^{type}$ from $\mathbb{M}_{\text{init}}^{type}$ for all possible inputs.

Results

Utilizing the NuSMV model checker ver. 2.6 [83], we obtained the voltage sequences shown in Table 7.2 for implementing the 1-bit full adder sub-computation. The FD adder requires the least number of execution steps because no input data needs to be recovered by the end of the computation. However, the ND and SD adders only require one additional step, so the extra cost associated with maintaining some or all of the input data is minimal with respect to our designs.

In terms of required memory components, all three designs utilize two auxiliary work memristors to help reduce the number of execution steps. Additionally, a single memristor can be used for the carry-bit as this memristor is simply overwritten with the carry-out value at the end of each sub-computation. Since the ND design requires a reserved area of memory for the output, $3n + 3$ memristors are necessary to implement n-bit addition. In contrast, the SD and FD n-bit adders need only encode the input data, so $2n + 3$ memristors suffice for their implementation.

When compared to other in-memory serial adders in the literature (see Table 6.1), we find that the semi-destructive and fully-destructive designs offer the greatest reduction in the required number

Time Step	$\mathbb{W}^{\text{ND}}$	$\mathbb{W}^{\text{SD}}$	$\mathbb{W}^{\text{FD}}$
1	$(Z, V_{HI}, V_{HI}, V_{LOW}, V_{HI}, V_{LOW})$	$(V_{HI}, Z, V_{HI}, V_{HI}, V_{HI})$	$(V_{HI}, Z, Z, V_{HI}, V_{HI})$
2	$(Z, V_{HI}, Z, V_{HI}, V_{HI}, V_{HI})$	$(V_{HI}, Z, Z, Z, V_{HI})$	$(Z, V_{HI}, Z, V_{HI}, V_{HI})$
3	$(Z, V_{LOW}, V_{HI}, V_{HI}, V_{HI}, V_{HI})$	$(V_{LOW}, Z, V_{LOW}, Z, V_{HI})$	$(V_{LOW}, Z, V_{HI}, V_{LOW}, Z)$
4	$(Z, V_{HI}, V_{LOW}, Z, V_{HI}, Z)$	$(Z, V_{HI}, V_{HI}, V_{HI}, Z)$	$(Z, V_{HI}, Z, V_{HI}, Z)$
5	$(V_{HI}, Z, Z, Z, V_{LOW}, V_{LOW})$	$(Z, Z, V_{HI}, V_{HI}, V_{LOW})$	$(V_{LOW}, Z, V_{LOW}, V_{HI}, V_{LOW})$
6	$(V_{LOW}, Z, Z, V_{HI}, V_{HI}, V_{HI})$	$(Z, V_{LOW}, Z, V_{LOW}, V_{HI})$	$(V_{HI}, V_{LOW}, Z, V_{LOW}, V_{HI})$
7	$(V_{HI}, Z, V_{HI}, V_{LOW}, V_{HI}, Z)$	$(Z, Z, V_{LOW}, V_{HI}, V_{LOW})$	—

Table 7.2: Voltage sequences for implementing the non-destructive, semi-destructive, and fully-destructive 1-bit full adder.

Time Step	m_k^x	m_k^y	m^c	m_1^a	m_2^a	m_k^s
Init	x_k	y_k	c_{in}	0	0	0
1	x_k	$y_k \vee \neg c_{in}$	$\neg y_k \vee c_{in}$	0	$\neg y_k \wedge \neg c_{in}$	0
2	x_k	1	$\neg y_k \vee c_{in}$	$\neg y_k \wedge c_{in}$	$\neg y_k$	$\neg y_k \wedge c_{in}$
3	x_k	$y_k \wedge \neg c_{in}$	1	$y_k \oplus c_{in}$	$\neg y_k \vee \neg c_{in}$	$y_k \oplus c_{in}$
4	x_k	y_k	$y_k \wedge c_{in}$	$y_k \oplus c_{in}$	1	$y_k \oplus c_{in}$
5	1	y_k	$y_k \wedge c_{in}$	$y_k \oplus c_{in}$	$\neg x_k$	$\neg x_k \wedge (y_k \oplus c_{in})$
6	$x_k \wedge \neg (y_k \oplus c_{in})$	y_k	$y_k \wedge c_{in}$	$x_k \vee (y_k \oplus c_{in})$	$\neg x_k \vee \neg (y_k \oplus c_{in})$	$x_k \oplus y_k \oplus c_{in}$
7	x_k	y_k	$(x_k \wedge y_k) \vee (x_k \wedge c_{in}) \vee (y_k \wedge c_{in})$	$x_k \wedge (y_k \oplus c_{in})$	1	$x_k \oplus y_k \oplus c_{in}$

Table 7.3: Generalized memristor states at each stage of a non-destructive 1-bit full adder with respect to arbitrary inputs

of memory components, while the non-destructive design is consistent with the optimized implementation from [69]. All three adders also improve computation time through a reduction in the total number of execution steps compared current

Illustrative Execution of 1-bit Full Adder

To demonstrate the operation of the 1-bit full adder implementation, we consider an example case of non-destructive addition when $x_k = y_k = 1$ and $c_{\text{in}} = 0$. Therefore, we have $\mathbb{M}_{\text{init}}^{\text{ND}} =$

$(1, 1, 0, 0, 0, 0)$, and we expect to obtain $\mathbb{M}_{\text{final}}^{\text{ND}} = (1, 1, 1, *, *, 0)$ through the application of the input voltage sequence $\mathbb{W}^{\text{ND}}$ in Table 7.2. The time evolution of our computational structure while performing this addition is shown in Figures 7.2a-7.2h and is explained below.

At $t = 1$ (Fig. 7.2a), the circuit has just been initialized. The values for x_k, y_k, and c_{in} are already stored in their respective memristors m_k^x, m_k^y, and m^c. The remaining memristors are switched to OFF. So, only m_k^x and m_k^y are in the ON state, and $u = 1$ because V_{HI} is being applied to m_k^y. Consequently, m^c and m_2^a retain the OFF state despite an input of V_{HI} because $V_{HI} - V_u < V_{ON}$. Therefore, the memory configuration is unaltered after applying $\mathbb{W}_1^{\text{ND}}$. At $t = 2$ (Fig. 7.2b), an identical result occurs when applying $\mathbb{W}_2^{\text{ND}}$. In particular, m_k^y is still in the ON state and has V_{HI} as input. Hence, $u = 1$ and the inequality $V_{HI} - V_u < V_{ON}$ holds, so memristors m_1^a, m_2^a, and m_k^s retain their OFF state. Similarly, m_k^x and m^c retain their respective states of ON and OFF, respectively, because the input Z acts as an identity function. At $t = 3$ (Fig. 7.2c), m_k^y is in the ON state with corresponding input V_{LOW}, so $u = 0$ by definition. Consequently, m_k^y will remain in the ON state because $V_{LOW} - V_u > V_{OFF}$. However, it will also be the case that $V_{HI} - V_u > V_{ON}$. So, the memristors m^c, m_1^a, m_2^a, and m_k^s will all switch to ON. At $t = 4$ (Fig. 7.2d), both m_k^y and m_2^a are in the ON state with input V_{HI}, so $u = 1$. However, m^c is in the ON state with input V_{LOW}, and $V_{LOW} - V_u < V_{OFF}$. So, m^c will switch to OFF, but the remaining memristors will all retain their respective states. At $t = 5$ (Fig. 7.2e), we again have $u = 1$ because m_k^x is ON with V_{HI} as input. Since m_2^a and m_k^s both have V_{LOW} as input and are in the ON state, these two memristors will switch to OFF while all others retain their state. At $t = 6$ (Fig. 7.2f), V_{LOW} is now being applied to m_k^x, but it is still the case that $u = 1$ because m_1^a is in the ON state with V_{HI} as input. Hence, m_2^a and m_k^s will retain their state and m_k^x will switch to OFF. At $t = 7$ (Fig. 7.2g), there are no memristors in the ON state with a corresponding input of V_{HI}, so $u = 0$ by definition. Hence, $V_{LOW} - V_u > V_{OFF}$ and $V_{HI} - V_u > V_{ON}$. Therefore, memristors m_k^x, m^c, and m_2^a will all switch to ON while the remaining components retain their respective states. The computation

is then completed at $t = 8$ (Fig. 7.2h) with $\mathbb{M}_8^{\text{ND}} = (1, 1, 1, 1, 1, 0)$, which is consistent with the desired output $\mathbb{M}_{\text{final}}^{\text{ND}} = (1, 1, 1, *, *, 0)$.

The sequence of memristor state evolutions for arbitrary x_k, y_k, and c_{in} is shown in Table 7.3. In particular, the memristors m_k^x and m_k^y used to store the input data are utilized as work units in that they can be overwritten during the procedure. However, the applied voltage sequence is structured in such a way that the data is recovered by the end of the computation.

Conclusion and Future Work

We discover new one-bit adder designs using bounded model checking on the logical dynamics of memristors without explicitly reducing addition to more fundamental operations such as IMPLY logic. We present compact design solutions for n-bit in-memory addition using varying degrees of destructive operations. Our non-destructive, semi-destructive, and fully-destructive adders are shown to be state-of-the-art in the number of memristors required for computation.

Several directions for future research remain open. The automated design of n-bit multipliers that compare favorably to other approaches is an immediate next step. Using learning methods to explore different non-linear topologies of memristors may also lead to interesting non-intuitive compact, fast and energy-efficient implementations of Boolean formula.

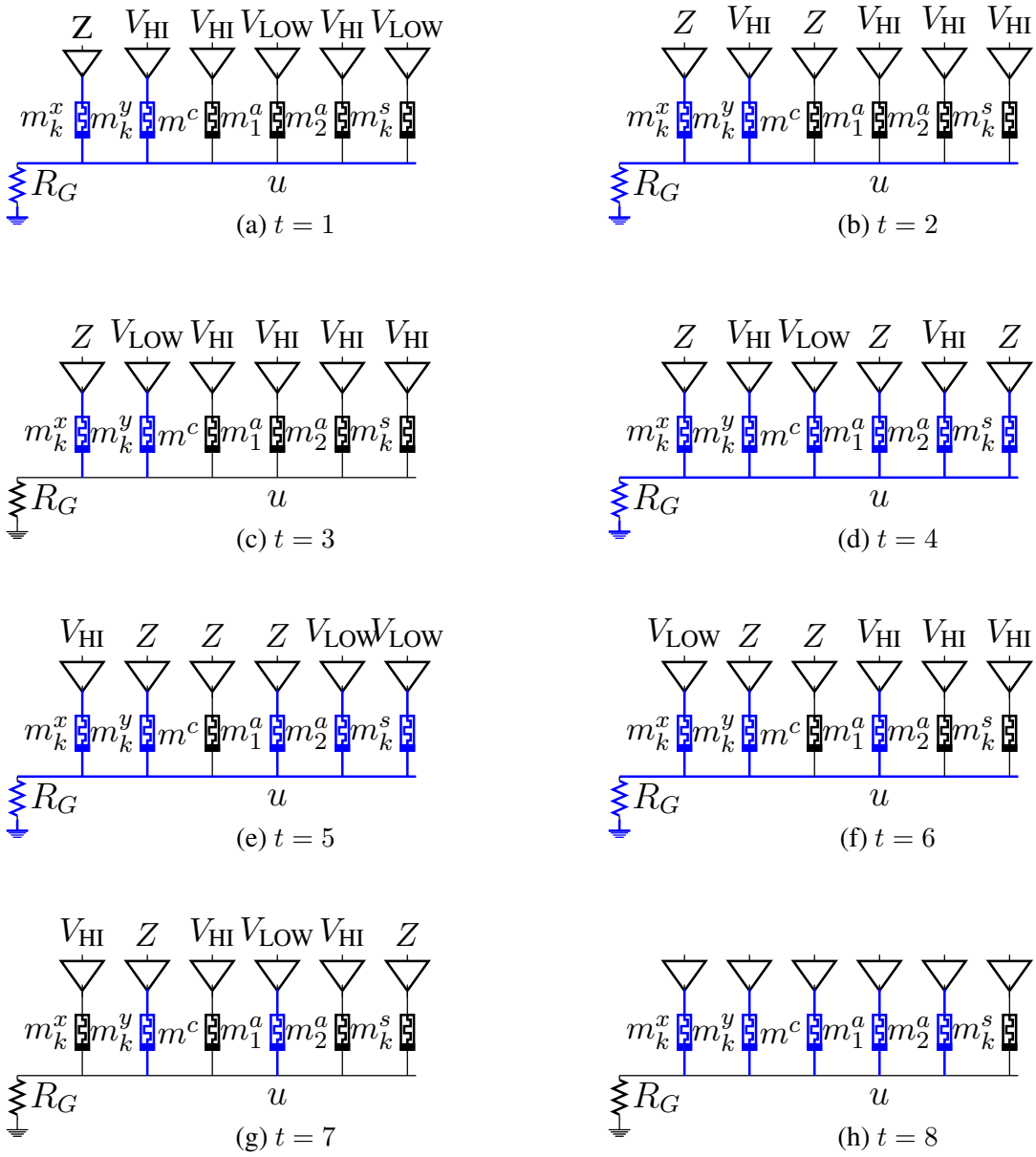

Z V_{HI} V_{HI} V_{LOW} V_{HI} V_{LOW}
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(a) $t = 1$
Z V_{HI} Z V_{HI} V_{HI} V_{HI}
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(b) $t = 2$
Z V_{LOW} V_{HI} V_{HI} V_{HI} V_{HI}
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(c) $t = 3$
Z V_{HI} V_{LOW} Z V_{HI} Z
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(d) $t = 4$
V_{HI} Z Z Z V_{LOW} V_{LOW}
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(e) $t = 5$
V_{LOW} Z Z V_{HI} V_{HI} V_{HI}
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(f) $t = 6$
V_{HI} Z V_{HI} V_{LOW} V_{HI} Z
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(g) $t = 7$
m_k^x m_k^y m^c m_1^a m_2^a m_k^s
R_G u
(h) $t = 8$

Figure 7.2: Memory state at each stage of a 1-bit full adder procedure for non-destructive (ND) addition with inputs $x_k = y_k = 1$ and $c_{in} = 0$. The blue wire (memristor) denotes $u = 1$ ($m = 1$).

CHAPTER 8: BOOLEAN MATRIX MULTIPLICATION WITH 3D CROSSBARS

In this chapter, we focus on a computation-in-memory solution to the problem of multiplying a set of Boolean matrices, also known as Boolean matrix chain multiplication (BMCM). This is a fundamental computational task with applications in graph theory, group testing, data compression, and digital signal processing. In particular, we propose a framework for mapping arbitrary instances of BMCM to a 3-dimensional (3D) crossbar memory architecture consisting of unidirectional 1-diode 1-resistor (1D1R) structures.

As we have seen, traditional Resistive Random Access Memory (ReRAM) architectures are crossbars, or cross-point memories, consisting of two sets of parallel wires, with each wire from one set placed perpendicularly to every wire in the other set. At every junction of wires, there is an interconnect acting as a switch between two wires. These interconnects typically consist of 1-transistor 1-resistor (1T1R) or 1-diode 1-resistor (1D1R) structures. However, it has been shown that in the context of 3-dimensional memory stacking, the usage of 1D1R cells is a more effective [90] and cost-efficient solution [91] than 1T1R. These 3D ReRAM memories are simply crossbars with more than one layer of 1D1R interconnects. An example can be seen in Figure 8.1. An abstraction of these memories is captured by Definition 7.

Definition 7. 3D CROSSBAR *An $|R| \times |C| \times L$ 3D crossbar is a 3-tuple $\mathcal{X} = (M, R, C)$ where*

- $M = \{M^1, \ldots, M^L\}$ *is a set of $|R| \times |C|$ Boolean matrices, where*
  $M^k = \begin{pmatrix} M^k_{11} & M^k_{12} & \ldots & M^k_{1|C|} \\ \vdots & \vdots & \ddots & \vdots \\ M^k_{|R|1} & M^k_{|R|2} & \ldots & M^k_{|R||C|} \end{pmatrix}$ *represents a matrix of interconnects with $|R|$ rows and $|C|$ columns. Each $m^k_{ij} \in \{0, 1\}$ denotes the state of the device connecting R_i with C_j*

in layer k.

- $R = \{R^1, \ldots, R^{L_R}\}$ *is the set of row wire vectors, where* $R^k = \{R^k_1, \ldots, R^k_{|R|}\}$ *and* $r^k_i \in \{0, 1\}$ *provides the same input voltage to every interconnect in* R_i *of layer* $2k - 1$.

- $C = \{C^1, \ldots, C^{L_C}\}$ *is the set of column wire vectors, where* $C^k = \{C^k_1, \ldots, C^k_{|C|}\}$ *and* $c^k_j \in \{0, 1\}$ *provides the same input voltage to every interconnect in column* j *of layer* $2k$.

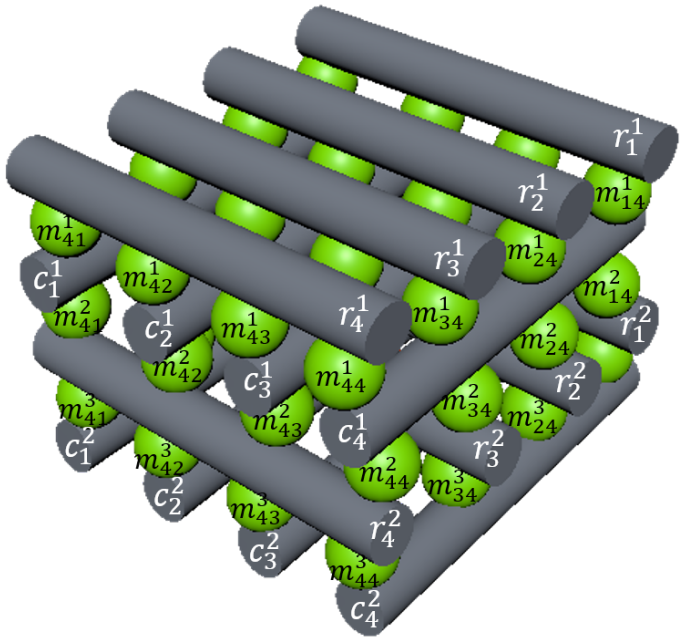


Figure 8.1: $4 \times 4 \times 3$ 3D crossbar. There are two sets of row and column wires and three layers of interconnects.

Since all of the components in a 3D crossbar are unidirectional, they follow the unidirectional flow axiom that we discussed in the previous chapter. We redefine it in Axiom 4 within the context of 3D crossbars. For the rest of this chapter, we assume square matrices for the sake of simplicity.

Axiom 4 (Unidirectional Flow)**.** *Let* $\mathcal{X} = (M, R, C)$ *be an* $n \times n \times L$ *crossbar. Then* $\forall i, j, k; 1 \leq i, j \leq n; 1 \leq k \leq L - 1$:

$$\left[\left(r_i^k \wedge m_{ij}^{2k-1}\right) \implies c_j^k\right] \wedge \left[\left(c_j^k \wedge m_{ij}^{2k}\right) \implies r_i^{k+1}\right]$$

While the problem of performing logic computations using crossbars has been studied in the literature [51, 78, 79, 92, 93, 1, 69, 82], leveraging the structure of a 3-dimensional crossbar for computation has largely gone unexplored. In this chapter, we take a step in this direction by computing Boolean matrix chain products within 3D ReRAM. We use the terms 3D ReRAM and crossbar interchangeably.

The Boolean matrix multiplication (BMM) problem may be defined as follows. Given two Boolean matrices $X^1 = (x_{ij}^1) \in \{0,1\}^{n\times n}$ and $X^2 = (x_{ij}^2) \in \{0,1\}^{n\times n}$, we wish to compute their product $S = X^1X^2 = (s_{ij})$, where $s_{ij} = \bigvee_{k=1}^{n} (x_{ik}^1 \wedge x_{kj}^2)$.

$$X^1 = \begin{pmatrix} x_{11}^1 & \dots & x_{1n}^1 \\ \vdots & \ddots & \vdots \\ x_{n1}^1 & \dots & x_{nn}^2 \end{pmatrix}, \quad X^2 = \begin{pmatrix} x_{11}^2 & \dots & x_{1n}^2 \\ \vdots & \ddots & \vdots \\ x_{n1}^2 & \dots & x_{nn}^2 \end{pmatrix}$$

$$S = \begin{pmatrix} \bigvee_{i=1}^{n} (x_{1i}^1 \wedge x_{i1}^2) & \bigvee_{i=1}^{n} (x_{1i}^1 \wedge x_{i2}^2) & \dots & \bigvee_{i=1}^{n} (x_{1i}^1 \wedge x_{in}^2) \\ \bigvee_{i=1}^{n} (x_{2i}^1 \wedge x_{i1}^2) & \bigvee_{i=1}^{n} (x_{2i}^1 \wedge x_{i2}^2) & \dots & \bigvee_{i=1}^{n} (x_{2i}^1 \wedge x_{in}^2) \\ \vdots & \vdots & \ddots & \vdots \\ \bigvee_{i=1}^{n} (x_{ni}^1 \wedge x_{i1}^2) & \bigvee_{i=1}^{n} (x_{ni}^1 \wedge x_{i2}^2) & \dots & \bigvee_{i=1}^{n} (x_{ni}^1 \wedge x_{in}^2) \end{pmatrix}$$

Given Boolean matrices $X^1, \dots, X^\alpha$, where $X^k = (x_{ij}^k) \in \{0,1\}^{n\times n}$, we define the k-chain product of these matrices by $S^k = (s_{ij}^k) = X^1X^2 \cdots X^{k+1}$ as defined by equation (8.3).

Methodology

The basis of our method consists of redirecting the flow of information through the crossbar based on the values of its interconnects. In the case of an electrical system, this can be achieved by applying a high voltage with respect to ground on some row wires in the first layer of the crossbar, grounding all of the bottommost wires in said crossbar, and configuring the interconnects based on variables of the formula ϕ that we wish to compute in such a way that electrical current will flow into the grounded wires if and only if $\phi = 1$. Thus, in principle this method follows the paths-based logic paradigm. In the context of our abstraction, this means that, given a 3D crossbar $\mathcal{X} = (M, \{R^1, \ldots, R^{L_R}\}, \{C^1, \ldots, C^{L_C}\})$, a voltage is applied at some row wires $r^1_{i_1} = r^1_{i_2} = \cdots = r^1_{i_k} = 1$. This will generate a flow of current such that, by successively applying Axiom 4, will result in some rows $r^{L_R}_{i_1} = r^{L_R}_{i_2} = \cdots = r^{L_R}_{i_k} = 1$ having flow as well (or $c^{L_C}_{i_1} = c^{L_C}_{i_2} = \cdots = c^{L_C}_{i_k} = 1$).

For any layer k, we can define the values of r^k_i and c^k_j as specified in (8.1) and (8.2), respectively.

$$r^k_i = \bigvee_{j=1}^{n} \left(c^{k-1}_j \wedge m^{2(k-1)}_{ij} \right), r^1_i \text{ is the } i^{th} \text{ input} \tag{8.1}$$

$$c^k_j = \bigvee_{i=1}^{n} \left(r^k_i \wedge m^{2k-1}_{ij} \right) \tag{8.2}$$

$$s^k_{ij} = \bigvee_{t=1}^{n} \left(s^{k-1}_{it} \wedge x^{k+1}_{tj} \right), s^1_{ij} = \bigvee_{t=1}^{n} \left(x^1_{it} \wedge x^2_{tj} \right) \tag{8.3}$$

We can expand equations (8.1) and (8.2) to obtain (8.4) and (8.5). Note the striking similarity between the structure of these two formulas and (8.6), which corresponds to the expansion of equation (8.3). Recall that (8.3) is an arbitrary entry in the matrix corresponding the product of

some Boolean matrices $X^1, \ldots, X^{k+1}$. This similarity provides some intuition as to the feasibility of mapping the BMCM problem onto a 3-dimensional crossbar. In fact, the mapping is rather simple, as specified in Theorem 7. The crux of the procedure lies in configuring the layers of interconnects according to (8.7). A pictorial representation can be seen in Figure 8.2.

$$r_i^k = \bigvee_{j_{k-1}=1}^{n} \left(\ldots \left(\bigvee_{j_2=1}^{n} \left(\bigvee_{i_2=1}^{n} \left(\bigvee_{j_1=1}^{n} \left(\bigvee_{i_1=1}^{n} \left(r_{i_1}^1 \wedge m_{i_1 j_1}^1 \right) \wedge m_{i_2 j_1}^2 \right) \wedge m_{i_2 j_2}^3 \right) \wedge m_{i_3 j_2}^4 \right) \ldots \right) \wedge m_{i j_{k-1}}^{2(k-1)} \right) \tag{8.4}$$

$$c_j^k = \bigvee_{i_k=1}^{n} \left(\ldots \left(\bigvee_{j_2=1}^{n} \left(\bigvee_{i_2=1}^{n} \left(\bigvee_{j_1=1}^{n} \left(\bigvee_{i_1=1}^{n} \left(r_{i_1}^1 \wedge m_{i_1 j_1}^1 \right) \wedge m_{i_2 j_1}^2 \right) \wedge m_{i_2 j_2}^3 \right) \wedge m_{i_3 j_2}^4 \right) \ldots \right) \wedge m_{i_k j}^{2k-1} \right) \tag{8.5}$$

$$s_{\gamma j}^k = \bigvee_{i=1}^{n} \left(\ldots \left(\bigvee_{j_2=1}^{n} \left(\bigvee_{i_2=1}^{n} \left(\bigvee_{j_1=1}^{n} \left(\bigvee_{i_1=1}^{n} \left(x_{\gamma i_1}^1 \wedge x_{i_1 j_1}^2 \right) \wedge x_{j_1 i_2}^3 \right) \wedge x_{i_2 j_2}^4 \right) \wedge x_{j_2 i_3}^5 \right) \ldots \right) \wedge x_{ij}^{k+1} \right) \tag{8.6}$$

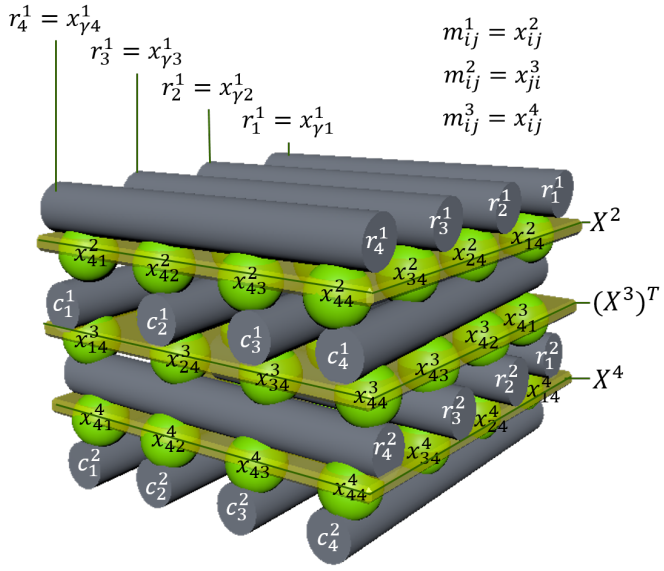


Figure 8.2: $4 \times 4 \times 3$ crossbar illustrating the configuration proposed in order to compute the product of Boolean matrices.

$$M^k = \begin{cases} X^{k+1} & , k \text{ odd} \\ (X^{k+1})^T & , k \text{ even} \end{cases} \tag{8.7}$$

Theorem 7. *Let $\mathcal{X} = (M, R, C)$ be an $n \times n \times L$ 3D crossbar and let $X^1, \ldots, X^\alpha, \alpha \geq 2$, denote a set of Boolean matrices with k-chain product $S^k = (s_{ij}^k) = X^1 X^2 \cdots X^{k+1}$. If $M^k = \begin{cases} X^{k+1} & , k \text{ odd} \\ (X^{k+1})^T & , k \text{ even} \end{cases}$, then $\begin{cases} c_j^{\alpha/2} = s_{\gamma j}^{\alpha-1} & , \alpha \text{ even} \\ r_i^{(\alpha+1)/2} = s_{\gamma i}^{\alpha-1} & , \alpha \text{ odd} \end{cases}$ for any row index $\gamma \in \{1, \ldots, n\}$.*

Proof. Let $r_i^1 = x_{\gamma i}^1$. The proof is by induction on α, where the notations $=_{(i)}$ and $=_{IH}$ indicate that the result follows from equation (i) and the inductive hypothesis, respectively.

Base case: When $\alpha = 2$,

$$c_j^{\alpha/2} = c_j^1 =_{(8.2)} \bigvee_{i=1}^{n} (r_i^1 \wedge x_{ij}^2) = \bigvee_{i=1}^{n} (x_{\gamma i}^1 \wedge x_{ij}^2) =_{(8.3)} s_{\gamma j}^1$$

Inductive hypothesis: Assume that $c_j^{\beta/2} = s_{\gamma j}^{\beta-1}$ for even β and $r_i^{(\beta+1)/2} = s_{\gamma i}^{\beta-1}$ for odd β.

Inductive step: For $\alpha = \beta + 1 > 2$, two cases arise.

- $\beta + 1$ even:

$$\begin{aligned} c_j^{\alpha/2} = c_j^{(\beta+1)/2} &=_{(8.2)} \bigvee_{i=1}^{n} \left(r_i^{(\beta+1)/2} \wedge m_{ij}^{\beta} \right) \\ &=_{(8.1)} \bigvee_{i=1}^{n} \left(\bigvee_{k=1}^{n} \left(c_k^{(\beta-1)/2} \wedge m_{ik}^{\beta-1} \right) \wedge m_{ij}^{\beta} \right) \\ &=_{(8.7)} \bigvee_{i=1}^{n} \left(\bigvee_{k=1}^{n} \left(c_k^{(\beta-1)/2} \wedge x_{ki}^{\beta} \right) \wedge x_{ij}^{\beta+1} \right) \\ &=_{IH} \bigvee_{i=1}^{n} \left(\bigvee_{k=1}^{n} \left(s_{\gamma k}^{\beta-2} \wedge x_{ki}^{\beta} \right) \wedge x_{ij}^{\beta+1} \right) \\ &=_{(8.3)} \bigvee_{i=1}^{n} \left(s_{\gamma i}^{\beta-1} \wedge x_{ij}^{\beta+1} \right) =_{(8.3)} s_{\gamma j}^{\beta} \end{aligned}$$

- $\beta + 1$ odd:

$$\begin{aligned} r_i^{(\alpha+1)/2} &= r_i^{(\beta+2)/2} \\ &=_{(8.1)} \bigvee_{j=1}^{n} \left(c_j^{\beta/2} \wedge m_{ij}^{\beta} \right) \\ &=_{(8.2)} \bigvee_{j=1}^{n} \left(\bigvee_{k=1}^{n} \left(r_k^{\beta/2} \wedge m_{kj}^{\beta-1} \right) \wedge m_{ij}^{\beta} \right) \\ &=_{(8.7)} \bigvee_{j=1}^{n} \left(\bigvee_{k=1}^{n} \left(r_k^{\beta/2} \wedge x_{kj}^{\beta} \right) \wedge x_{ji}^{\beta+1} \right) \\ &=_{IH} \bigvee_{j=1}^{n} \left(\bigvee_{k=1}^{n} \left(s_{\gamma k}^{\beta-2} \wedge x_{kj}^{\beta} \right) \wedge x_{ji}^{\beta+1} \right) \\ &=_{(8.3)} \bigvee_{j=1}^{n} \left(s_{\gamma j}^{\beta-1} \wedge x_{ji}^{\beta+1} \right) =_{(8.3)} s_{\gamma i}^{\beta} \qquad \square \end{aligned}$$

By applying Theorem 7 on $X^1, \ldots, X^\alpha \in \{0,1\}^{n \times n}$ for some row index γ, we can compute the entries in the γ^{th} row vector of $S^{\alpha-1}$. That is, the values of $s_{\gamma 1}^{\alpha-1}, s_{\gamma 2}^{\alpha-1}, \ldots, s_{\gamma n}^{\alpha-1}$ will be contained in $r_1^{(\alpha+1)/2}, r_2^{(\alpha+1)/2}, \ldots, r_n^{(\alpha+1)/2}$ when α is odd and in $c_1^{\alpha/2}, c_2^{\alpha/2}, \ldots, c_n^{\alpha/2}$ when α is even. Therefore, we need only repeat this procedure n times, once for each $\gamma \in \{1, \ldots, n\}$, in order to compute

$S^{\alpha-1}$. We elucidate this approach with a simple example. Let $X^1, X^2, X^3 \in \{0,1\}^{4\times 4}$.

$$X^1 = X^2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad X^3 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Clearly, $S^2 = X^1X^2X^3 = X^3$ since X^1 and X^2 are identity matrices. This problem instance can be mapped to a $4\times4\times2$ crossbar $\mathcal{X} = (\{M^1, M^2\}, \{R^1, R^2, R^3, R^4\}, \{C^1, C^2, C^3, C^4\})$ by setting $M^1 = X^2$ and $M^2 = (X^3)^T$. In order to compute $s^2_{11}, s^2_{12}, s^2_{13}, s^2_{14}$, let $r^1_1 = 1, r^1_2 = r^1_3 = r^1_4 = 0$. From Theorem 7, it follows that $r^2_1, r^2_2, r^2_3, r^2_4$ will hold the values of $s^2_{11}, s^2_{12}, s^2_{13}, s^2_{14}$. Similarly, the values for the second row of S^2 (i.e. $s^2_{21}, s^2_{22}, s^2_{23}, s^2_{24}$) are computed by setting $r^1_2 = 1, r^1_1 = r^1_3 = r^1_4 = 0$ as can be seen in Figure 8.3.

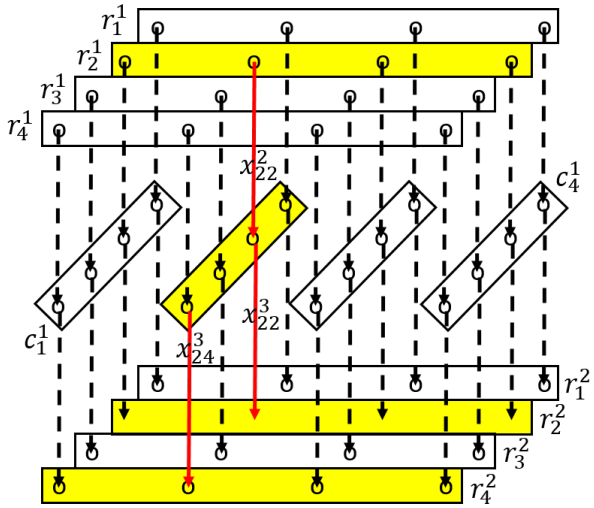


Figure 8.3: Dashed lines represent interconnections between row and column wires. Yellow bars denote a wire with a truth value of 1 and solid red lines are due to Axiom 4; they correspond to interconnects redirecting flow from one wire to another.

Multiplying two $n \times n$ Boolean matrices has been shown to be computable in $O(n^3/p + \log(p/n^2))$ time on a hypercube with $n^2 \leq p \leq n^3$ processors. This leads to the fastest runtime of $O(n)$ with n^2 processors. Using this processor array model of parallel computation, we utilize n^2 processing elements in order to solve the Boolean matrix multiplication problem in linear time on a 2D crossbar. In this case, each memristive device can be viewed as a processing element. Thus, the proposed approach has an optimal runtime when we are multiplying two matrices. We conjecture that this optimality holds for computing the product of an arbitrary number of matrices.

Experimental Results

The preceding example has been verified via HSPICE simulations using Schottky diodes and resistors in 1D1R structures. Interconnects corresponding to variables with value 1 have LRS resistances of 10Ω and variables with value 0 have HRS resistances of $100\text{k}\Omega$. A $2V$ voltage pulse was applied on the topmost row wires in accordance with Theorem 7 and a resistor-to-ground with resistance $1M\Omega$ was placed on each of the bottommost wires in order to read the outputs, which are shown in the matrix R_V below. Note that the entries of R_V coincide with the entries of $S^2 = X^1X^2X^3 = X^3$ as intended. It can be seen that whenever some s_{ij}^2 evaluates to 1, a voltage value (with respect to ground) of $6 - 8mV$ is read. Conversely, whenever s_{ij}^2 evaluates to 0, a voltage value of approximately $6\mu V$ is measured at the outputs. We have also verified our approach on randomized Boolean matrices using a varying number of layers. While there is a significant read margin between 0 and 1 values when up to 4 layers are used, a substantial signal degradation is

observed for 3D crossbars with 8 or more layers.

$$R_V = \begin{pmatrix} 6.4374mV & 6.1352\mu V & 6.2094\mu V & 8.4733mV \\ 6.5884\mu V & 8.4738mV & 6.7009\mu V & 8.4738mV \\ 6.4275mV & 6.3153\mu V & 8.4734mV & 6.0886\mu V \\ 6.4288mV & 8.4746mV & 8.4747mV & 7.8898\mu V \end{pmatrix}$$

In order to compare the performance of 3D ReRAM against other popular memory architectures, we have simulated a 2 MB memory using the DESTINY memory modeling tool [5]. The results can be seen in Table 8.1. Note that each memory architecture prevails in some aspect of performance. eDRAM has the lowest read and write latency, SRAM requires a low write energy at the cost of area and substantial leakage power, 2D ReRAM has low leakage and read energy, and 3D ReRAM benefits from exceptionally small area. In fact, DESTINY demonstrates that a 32 GB memory would only occupy an area of 33.763 mm^2 when using 22 nm technology in a 16-layer 3D ReRAM.

Table 8.1: DESTINY [5] simulations of different memory architectures using 22 nm technology, including the 3D ReRAM used in this paper. In order to avoid bias, we utilize the default parameters included in the simulator. Simulation files can be found in eecs.ucf.edu/∼velasquez/Table7.1.

	Read Latency (ns)	Write Latency (ns)	Read Energy (pJ)	Write Energy (pJ)	Leakage (mW)	Area (mm^2)
3D ReRAM	124.09	139.51	122.33	129.11	9.852	0.0027
2D ReRAM	14.325	22.9	41.179	21.132	2.268	0.0365
SRAM	41.541	41.541	386.6	2.31	1924	1.259
eDRAM	9.179	9.179	41.558	1009	2.637	0.2764

It is worth noting that the approach proposed in this paper is applicable to matrices whose dimensions are greater than those of the 3D crossbar. This result follows from seminal work found in

Cannon's thesis, where it is shown that matrix multiplication can be carried out by smaller sub-matrix block products [94].

Concluding Remarks

We have shown how to compute the product of a set of Boolean matrices by mapping said matrices to a 3-dimensional crossbar memory. The correctness of the proposed approach was proven mathematically and a simple example was given to elucidate its effectiveness, with a read margin of three orders of magnitude between the output voltages of 0 and 1 values. As with earlier chapters, this result attempts to ameliorate the divide between the traditional computation model of von Neumann architectures and the memory-processor integration paradigm in 3D ReRAM.

CHAPTER 9: TRANSITIVE CLOSURE WITHIN 2-LAYERED MEMORY

In the previous chapter, a framework for computing the product of a set of Boolean matrices was presented. Recall that $k-1$ layers of interconnects are required to compute the product of k matrices. Unfortunately, it is notoriously difficult to fabricate these 3-dimensional crossbars with many layers. The most popular such architecture is Intel's XPoint memory [19] which uses 2 layers. This motivates the need for a computing paradigm that may be used on such existing architectures. In this chapter, we introduce a new in-memory computing architecture that can compute the transitive closure of graphs using the natural parallel flow of information in 3D crossbars with two layers in a manner similar to the one discussed in the previous chapter.

The transitive closure problem is a well-studied graph theory problem with ubiquitous applications in many different areas. In the theoretical domain, a classic result of Skyum and Valiant demonstrates how problems computable by circuits of polynomial size can be reduced to the transitive closure operation via projections [95]. In more applied domains, this problem underpins many important procedures, such as resolving database queries [96], parsing grammars [30], determining data dependencies at compilation [97], as well as formal verification algorithms such as model checking [98]. A thorough treatment of transitive closure computation and its applications can be found in Nuutila's doctoral thesis [99]. Inspired by these applications and motivated by the difficulties of parallelizing transitive closure on John von Neumann architectures, we investigate if the inherently parallel flow of information in 3D crossbar arrays can be used to perform graph transitive closure in an efficient manner. The proposed architecture can be implemented using 3D crossbar architectures with two layers of 1-diode 1-resistor (1D1R) interconnects. Our proposed procedure has a runtime complexity of $\mathcal{O}(n^2)$ using $\mathcal{O}(n^2)$ devices. This compares favorably to efficient algorithms on John von Neumann architectures with a time complexity of $\mathcal{O}(n^3/p + n^2 \log p)$ on p processors [100], leading to a runtime of $\mathcal{O}(n^2 \log n^2)$ using $\mathcal{O}(n^2)$ processors.

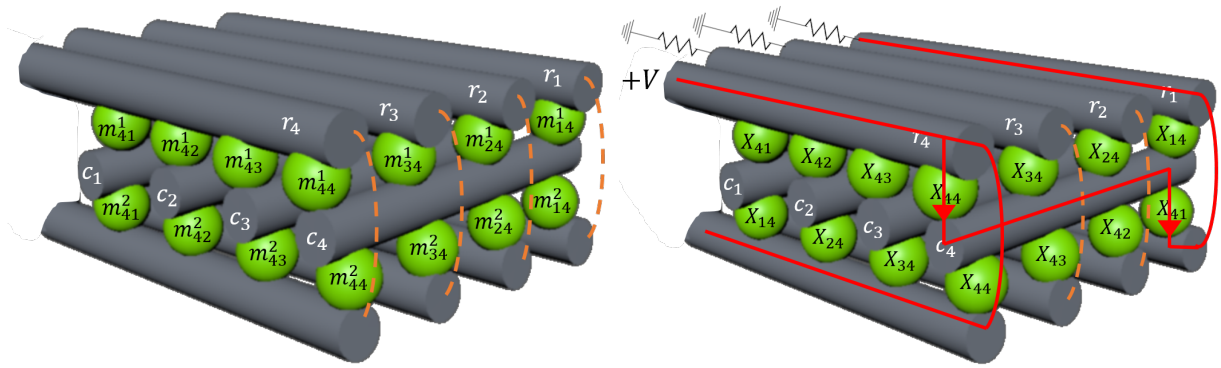


Figure 9.1: (*Left*) A graphical illustration of a 3D crossbar $\mathcal{X} = (M^1, M^2, R, C)$ with two layers of interconnects and external feedback loops. The dashed arcs denote an external interconnection between the corresponding bottom and top wires. (*Right*) Given $X = ((1,0,0,0),(0,1,0,0),(0,0,1,0),(1,0,0,1))$, we configure the crossbar in accordance with Theorem 8 so that $M^1 = X$ and $M^2 = X^T$. In order to compute the 4^{th} row vector of the transitive closure of X, a voltage bias is applied to R_4 and R_1, R_2, R_3 are grounded. Note that the values (r_1, r_2, r_3, r_4) obtained from the voltage readings of (R_1, R_2, R_3, R_4) will correspond to $(1,0,0,1)$, as expected.

The computational fabric we adopt in this paper is a 3D crossbar memory with two layers of 1-diode 1-resistor (1D1R) interconnects. In our approach, the top and bottom rows or wires of the crossbar are connected to each other using individual external connections as shown in Fig. 9.1. Definition 8 presents an abstraction of the 3D crossbar architecture. As in the previous chapter, we assume square matrices for simplicity.

Definition 8. *An $n \times n$ 3D crossbar with two externally interconnected layers is a 4-tuple $\mathcal{X} = (M^1, M^2, R, C)$, where*

- $M^k = \begin{pmatrix} M^k_{11} & M^k_{12} & \dots & M^k_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ M^k_{n1} & M^k_{n2} & \dots & M^k_{nn} \end{pmatrix}$ *is a Boolean matrix representing the interconnects with n rows and n columns. Each $m^k_{ij} \in \{0,1\}$ denotes the state of the device connecting R_i with C_j in layer $k \in \{1,2\}$. $m^k_{ij} = 1$ denotes a 1D1R device in the low-resistance state (LRS)*

state and $m_{ij}^k = 0$ denotes the same device in a high-resistance state (HRS).

- *$R = \{R_1, \ldots, R_n\}$ is a vector of row wire values and $r_i \in \{0, 1\}$ provides the same input voltage to every component M_{ij}^1 in the first layer. External feedback loops connect the bottommost row wires with the corresponding topmost wires. Hence, it is enough to model both the topmost and bottommost rows using the same vector of values. See Fig. 9.1 for an illustrative example.*

- *$C = \{C_1, \ldots, C_n\}$ is a vector of column wires and $c_j \in \{0, 1\}$ provides the same input to every component M_{ij}^2 in the second layer of the 3D crossbar.*

Since each 1D1R component contains a diode, current will only flow in one direction when said component is in the LRS state. Thus, the unidirectional flow described in the previous chapter holds. Without loss of generality, we assume that m_{ij}^1 and m_{ij}^2 allow current to flow only from wires R_i to C_j and C_j to R_i, respectively.

In order to frame the dynamics of 3D crossbars within the framework of a well-studied computational model, we represent the foregoing dynamics as a Boolean circuit. In this Boolean circuit, the value of each wire r_i, c_j is denoted by a Boolean function g_i^r, g_j^c. Similarly, the outputs of components M_{ij}^1, M_{ij}^2 are defined by Boolean functions g_{ij}^1, g_{ij}^2. For the remainder of this section, we define these functions.

Suppose we have a 3D crossbar $\mathcal{X} = (M^1, M^2, R, C)$. In order to induce a flow of current through the crossbar, a voltage bias is applied to some row wire R_i and other wire(s) are grounded. The trajectory of current will depend on the configuration of the interconnect matrices M^1, M^2. Due to the unidirectional flow of current imposed by 1D1R interconnects, current will flow from wire R_i to C_j if m_{ij}^1 is in the LRS state and wire R_i has a high voltage with respect to ground (i.e. $m_{ij}^1 = 1$ and $r_i = 1$). Similarly, current will flow from C_j to R_i when $m_{ij}^2 = 1$ and $c_j = 1$. We can thus

define the outputs of components M^1_{ij} and M^2_{ij} by $g^1_{ij}(r_i, m^2_{ij}) = r_i \wedge m^2_{ij}$ and $g^2_{ij}(c_j, m^1_{ij}) = c_j \wedge m^1_{ij}$, respectively. We can now define the values of r_i and c_j by equations (9.1) and (9.2), where $\mathcal{I}_i$ is 1 if a voltage is applied to R_i and 0 if R_i is grounded. This behavior is well-captured by a cyclic Boolean circuit [101]. See Fig. 9.2 for a pictorial definition.

$$\begin{aligned} c_j &= g^c_j \left(g^1_{1j}(r_1, m^1_{1j}), \ldots, g^1_{nj}(r_n, m^1_{nj}) \right) \\ &= \bigvee_{i=1}^{n} \left(r_i \wedge m^1_{ij} \right) \end{aligned} \tag{9.1}$$

$$\begin{aligned} r_i &= g^r_i \left(\mathcal{I}_i, g^2_{i1}(c_1, m^2_{i1}), \ldots, g^2_{in}(c_n, m^2_{in}) \right) \\ &= \mathcal{I}_i \vee \bigvee_{j=1}^{n} \left(c_j \wedge m^2_{ij} \right) \end{aligned} \tag{9.2}$$

In order to capture the dynamics of the feedback loop, we model the flow of information in the crossbar through a discrete notion of time. Let $r_{i,t}$ and $c_{j,t}$ denote the state of wires R_i and C_j during the t^{th} iteration of the feedback loop. At time $t = 0$, we have $r_{i,0} = \mathcal{I}_i$. The values of the wires then evolve from $t = 1$ onwards according to equations (9.3) and (9.4).

$$c_{j,t} = \bigvee_{i=1}^{n} \left(r_{i,t-1} \wedge m^1_{ij} \right) \tag{9.3}$$

$$r_{i,t} = \bigvee_{j=1}^{n} \left(c_{j,t} \wedge m^2_{ij} \right) \tag{9.4}$$

Equation (9.3) suggests that column j has a flow of current if and only if some R_i has a flow at time $t-1$ and the corresponding component M^1_{ij} is in the LRS state in the first layer of the 3D crossbar. Similarly, equation (9.4) describes the fact that wire R_i has a new flow of current at time t if and only if some wire C_j has a flow at time t and the corresponding component M^2_{ij} is in the LRS state

in the second layer. While the state of the flows in the wires changes as time evolves, the states of the interconnects are set at time $t = 0$ and do not evolve. Essentially, the interconnects store the data values in the 3D crossbar while the interconnecting wires enable the desired computation in our approach. This is in tune with the paths-based logic paradigm.

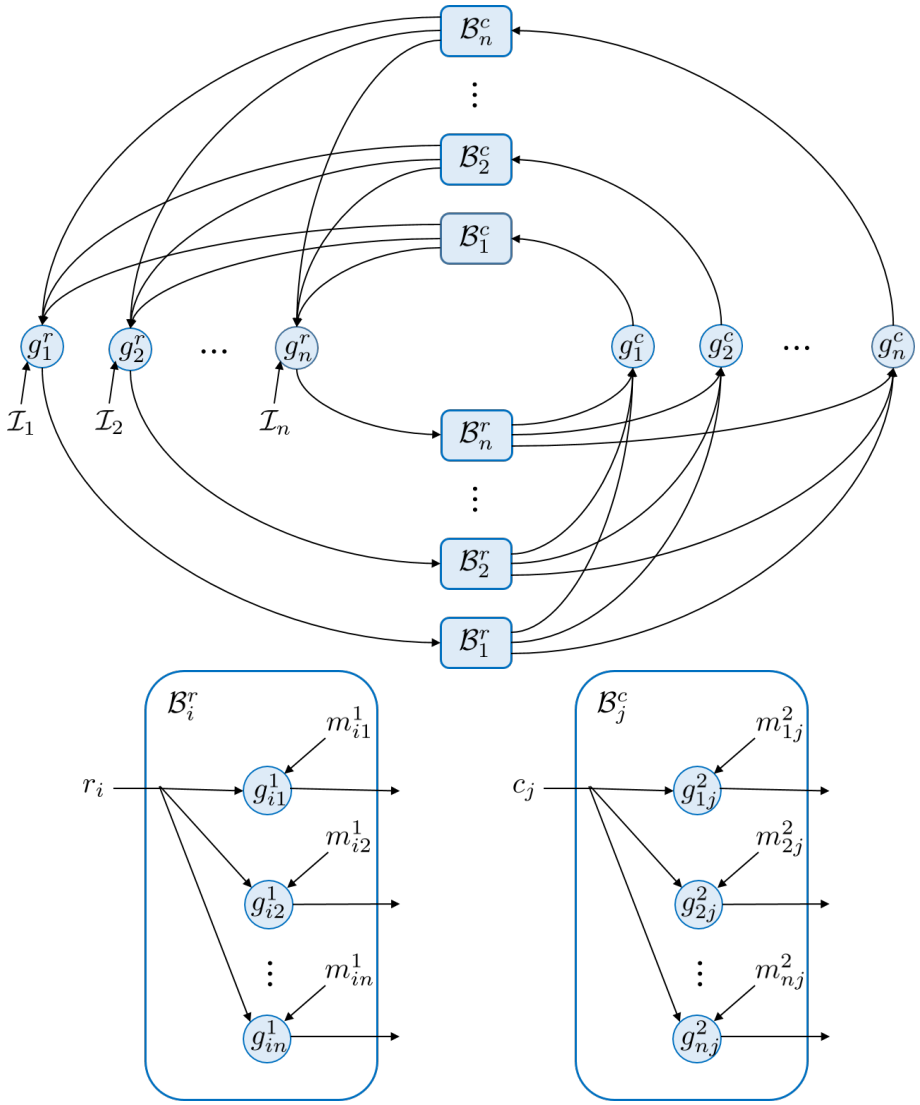


Figure 9.2: Cyclic Boolean circuit for a given $n \times n$ crossbar $\mathcal{X} = (M^1, M^2, R, C)$. In this circuit, each wire value r_i, c_j has a corresponding Boolean function $g_i^r : \{0,1\}^{n+1} \mapsto \{0,1\}$, $g_j^c : \{0,1\}^n \mapsto \{0,1\}$ which is a logical disjunction of its inputs. Similarly, $g_{ij}^1, g_{ij}^2 : \{0,1\}^2 \mapsto \{0,1\}$ are 2-bit conjunctions corresponding to the logical output of each m_{ij}^1 and m_{ij}^2, respectively.

Methodology

We are concerned with solving the transitive closure problem through repeated Boolean matrix multiplication using in-memory computing. The mapping proposed may be suitable for memories similar to Intel's recently unveiled 3D XPoint™ memory architecture [19], provided that feedback loops are added externally (see Fig. 9.1). Given a graph $G = (V, E)$, where V is the set of vertices in the graph and $E \in \{0,1\}^{|V|\times|V|}$ is its adjacency matrix, the transitive closure of G consists of the matrix X^*, where

$$X^*_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or there is a path from } v_i \text{ to } v_j \text{ in } G \\ 0 & \text{otherwise} \end{cases}$$

Let $X^{(k)}$ denote the k-reachability matrix specifying which nodes are reachable by a path of at most k edges. We assume that every node has a path of length 0 to itself, thus $X^{(0)} = I$, where I denotes the identity matrix. $X^{(k)}$ can be defined recursively as $X^{(k)} = (I \vee E)^k = X^{(k-1)}X^{(1)}$. Since the maximum length of a path in G is $|V| - 1$, then $X^* = X^{|V|-1} = (I \vee E)^{|V|-1}$. However, it is worth noting that a great deal of computations can be spared if the diameter of the graph is known. This follows from the fact that the transitive closure problem converges on a solution after $diam(G)$ (diameter of G) iterations. Thus, $X^* = (I \vee E)^{diam(G)}$. It has been observed that $diam(G) << |V|$ in real-world networks [6] (See Table 9.1).

The preceding argument implies that the transitive closure of a graph can be computed using repeated matrix multiplication. Theorem 8 below demonstrates how this procedure can be efficiently computed in an $n \times n$ crossbar with feedback loops. The values of the bits stored at the crosspoints are fixed during initialization and do not change for the entirety of the procedure. See Fig. 9.1 for a simple example.

Theorem 8. *Let $\mathcal{X} = (M^1, M^2, R, C)$ be an $n \times n$ 3D crossbar, X denote a Boolean matrix, I be the $n \times n$ identity matrix, and $\gamma \in \{1, \ldots, n\}$ be a row index . If $M^1 = X$, $M^2 = X^T$ and $(r_{10}, \ldots, r_{n0}) = (I_{\gamma 1}, \ldots, I_{\gamma n})$, then $r_{it} = (X^{2t})_{\gamma i}$ and $c_{jt} = (X^{2t-1})_{\gamma j}$ at any given time index t.*

Proof. The proof is by induction on the time index t.

(Base Case) $t = 1$:

$$c_{j1} = \bigvee_{k=1}^{n} \left(r_{k0} \wedge m^1_{kj}\right) \tag{9.5}$$

$$= \left(r_{\gamma 0} \wedge m^1_{\gamma j}\right) \vee \bigvee_{k \neq \gamma}^{n} \left(r_{k0} \wedge m^1_{kj}\right) \tag{9.6}$$

$$= r_{\gamma 0} \wedge m^1_{\gamma j} = m^1_{\gamma j} = X_{\gamma j} \tag{9.7}$$

Equation (9.5) states that the column wire C_{j1} has a flow if and only if a row wire has a flow and its corresponding interconnect in layer 1 is in the LRS state. This arises naturally from the design of our crossbar (see Eqn. (9.4)). Since the premise of the theorem sets wire values r_{i0} using a row vector of the identity matrix, all values r_{k0} such that $\gamma \neq k$ are set to 0. Equations (9.6) and (9.7) use this fact to algebraically simplify the first equation.

$$r_{i1} = \bigvee_{j=1}^{n} \left(c_{j1} \wedge m_{ij}^{2}\right) \tag{9.8}$$

$$= \bigvee_{j=1}^{n} \left(X_{\gamma j} \wedge m_{ij}^{2}\right) \tag{9.9}$$

$$= \bigvee_{j=1}^{n} \left(X_{\gamma j} \wedge X_{ji}\right) = \left(X^{2}\right)_{\gamma i} \tag{9.10}$$

Equation (9.8) captures the fact that the row wire gets a new flow if and only if one of the column wires gets a flow from equations (9.5) through (9.7). Equation (9.9) is the result of algebraically substituting equation (9.7) into equation (9.8). Equation (9.10) is derived using the fact that an interconnect in the second layer of the crossbar is in the LRS state if and only if the corresponding entry in the transpose of the matrix X is 1. The base case is established using equations (9.7) and (9.10).

(Inductive Hypothesis): Assume that $r_{it} = \left(X^{2t}\right)_{\gamma i}$ and $c_{it} = \left(X^{2t-1}\right)_{\gamma i}$ for all γ and all $i \leq t$.

(Induction): The value of the column wire $c_{j,t+1}$ can be computed as follows.

$$c_{j,t+1} = \bigvee_{k=1}^{n} \left(r_{kt} \wedge m_{kj}^{1}\right) = \bigvee_{k=1}^{n} \left(\left(X^{2t}\right)_{\gamma k} \wedge m_{kj}^{1}\right) \tag{9.11}$$

$$= \bigvee_{k=1}^{n} \left(\left(X^{2t}\right)_{\gamma k} \wedge X_{kj}\right) = \left(X^{2t+1}\right)_{\gamma j} \tag{9.12}$$

Equation (9.11) describes how the flow of current on a column arises from the flow of current on a row wire and a corresponding interconnect being in the LRS state. It also leverages the inductive hypothesis that $r_{kt} = \left(X^{2t}\right)_{\gamma k}$. Equation (9.12) exploits the fact that the first layer of the crossbar

stores the Boolean matrix X. Similarly, the value of the row wire $r_{i,t+1}$ can be computed as follows:

$$r_{i,t+1} = \bigvee_{j=1}^{n} \left(c_{j,t+1} \wedge m_{ij}^2\right) = \bigvee_{j=1}^{n} \left(\left(X^{2t+1}\right)_{\gamma j} \wedge m_{ij}^2\right) \tag{9.13}$$

$$= \bigvee_{j=1}^{n} \left(\left(X^{2t+1}\right)_{\gamma j} \wedge X_{ji}\right) = \left(X^{2t+2}\right)_{\gamma i} \tag{9.14}$$

Equation (9.13) uses the fact that there is a new flow on a row wire if and only if a column wire has flow and its corresponding component in the second layer is in the LRS state. It also leverages the result we proved in Equation (9.12) that $c_{j,t+1} = (X^{2(t+1)-1})_{\gamma j}$. Equation (9.13) is derived from the transpose of the Boolean matrix X being stored in the second layer of the 3D crossbar.

Hence, we have proved that $r_{it} = (X^{2t})_{\gamma i}$ and $c_{jt} = (X^{2t-1})_{\gamma j}$ by mathematical induction. □

Table 9.1: Number of nodes, edges, and diameter for all benchmark graphs exceeding one million nodes from the Stanford Large Network Data Collection. Note that the diameter of the networks is much smaller than their number of nodes as stated in [6].

Benchmark	Number of nodes	Number of edges	Diameter
roadNet-PA	1,088,092	1,541,898	786
com-Youtube	1,134,890	2,987,624	20
as-Skitter	1,696,415	11,095,298	25
roadNet-TX	1,379,917	1,921,660	1054
soc-Pokec	1,632,803	30,622,564	11
roadNet-CA	1,965,206	2,766,607	849
wiki-Talk	2,394,385	5,021,410	9
com-Orkut	3,072,441	117,185,083	9
cit-Patents	3,774,768	16,518,948	22
com-LiveJournal	3,997,962	34,681,189	17
soc-LiveJournal1	4,847,571	68,993,773	16
com-Friendster	65,608,366	1,806,067,135	32

From Theorem 8, we arrive at the transitive closure of a graph G using an in-memory computing architecture that implements repeated Boolean matrix multiplication. Repeating this procedure for each $\gamma \in \{1, \ldots, n\}$ yields the n vectors of the transitive closure matrix X^*. We illustrate this with the simple example below, where the matrices $X^{(1)}, \ldots, X^{(4)}$ are reachability matrices. Clearly, the transitive closure matrix is $X^* = X^{(4)}$. A visualization of how X^* is computed in a 5×5 crossbar and its cyclic Boolean circuit can be seen in Figs. 9.3 and 9.4.

$$X^{(1)} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad X^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$X^{(3)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad X^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

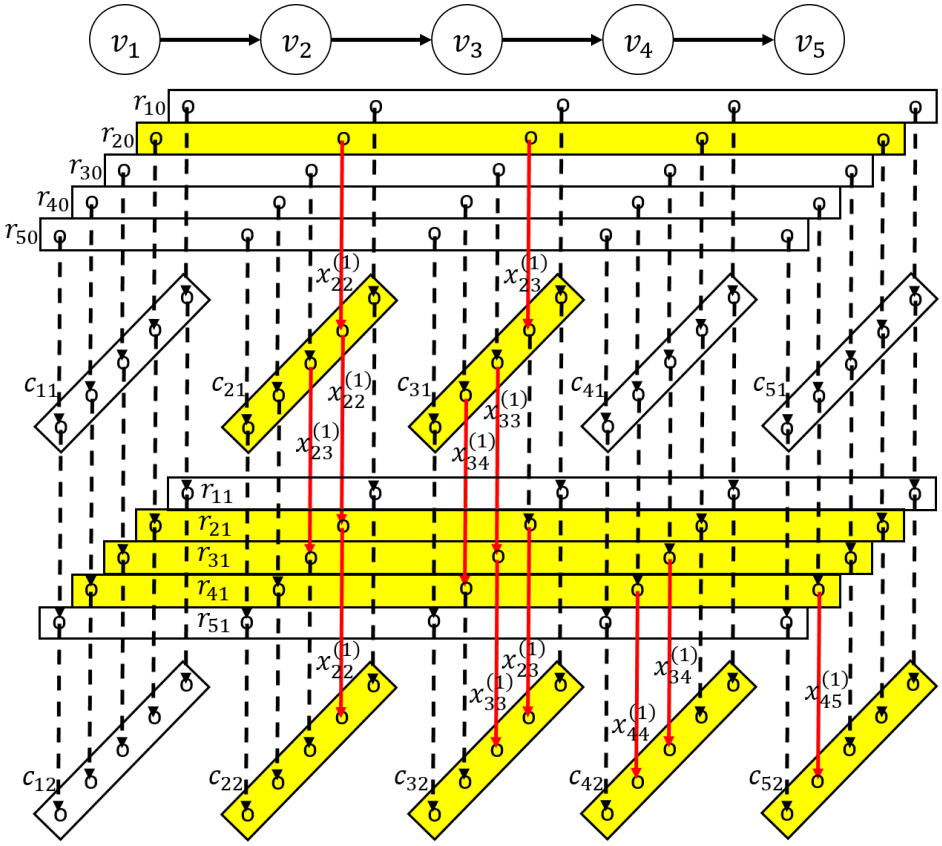


Figure 9.3: Procedure for computing the transitive closure X^* of directed graph $G = (\{v_1, v_2, v_3, v_4, v_5\}, E)$ (top). The two-layer crossbar with feedback loop $\mathcal{X} = (M^1, M^2, \{R_1, R_2, R_3, R_4, R_5\}, \{C_1, C_2, C_3, C_4, C_5\})$ (bottom) is unrolled in order to help visualize the computation. Dashed lines represent interconnections between row and column wires, yellow bars denote a wire with a value of 1 (i.e. current flows through it), and solid red lines correspond to interconnects redirecting the flow of current from one wire to another. In this case, we are computing the second row vector $(X^*_{21}, \ldots, X^*_{25})$ of X^* by setting $(r_{10}, r_{20}, r_{30}, r_{40}, r_{50}) = (0, 1, 0, 0, 0)$. Note that $(c_{12}, c_{22}, c_{32}, c_{42}, c_{52}) = (X^*_{21}, X^*_{22}, X^*_{23}, X^*_{24}, X^*_{25})$.

It follows from Theorem 8 that, given a 3D crossbar $\mathcal{X} = (M^1, M^2, R, C)$, $(r_1, \ldots, r_n)$ and $(c_1, \ldots, c_n)$ will converge on $(X^*_{\gamma 1}, \ldots, X^*_{\gamma n})$ when the wire vector $(r_{10}, \ldots, r_{n0})$ is configured according to $(I_{\gamma 1}, \ldots, I_{\gamma n})$, where I is the identity matrix. This convergence holds independently of timing errors, as is evidenced by equation (9.15) below, which an expansion of (9.4). This provides a definition for r_{it} that is independent of time.

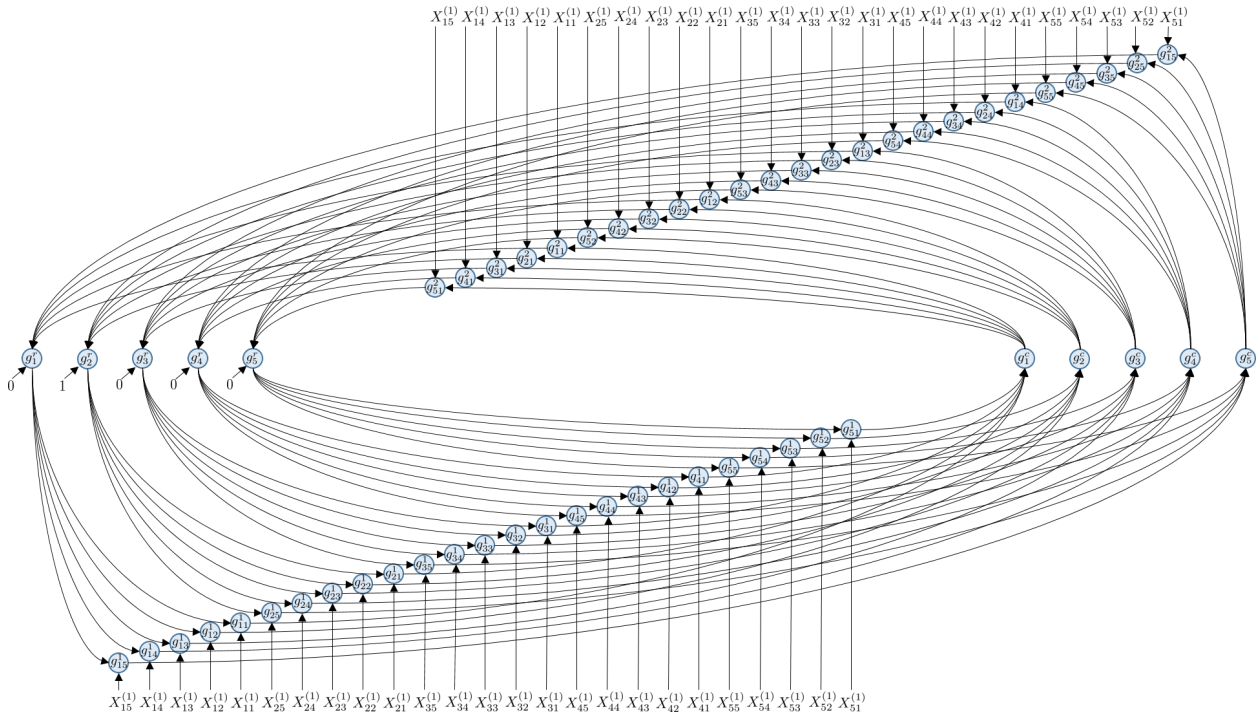


Figure 9.4: Boolean circuit corresponding to the example in Fig. 9.3. Boolean gates g_{ij}^1 and g_{ij}^2 have inputs $X_{ij}^{(1)}$ and $X_{ji}^{(1)}$ in accordance with Theorem 8. Each g_i^r has an input $\mathcal{I}_i$ corresponding to the value of r_{i0}.

$$
\begin{aligned}
r_{it} &= \bigvee_{j_1=1}^{n} \left(\bigvee_{i_1=1}^{n} \cdots \left(\bigvee_{j_t=1}^{n} \left(\bigvee_{i_t=1}^{n} (r_{i_t,0} \wedge m_{i_t j_t}^1) \cdots \right) \right) \right) \\
&= \bigvee_{j_1=1}^{n} \bigvee_{i_1=1}^{n} \cdots \bigvee_{j_t=1}^{n} \left(m_{\gamma j_t}^1 \wedge \cdots \wedge m_{i_1 j_1}^1 \wedge m_{i j_1}^2 \right) \qquad (9.15)
\end{aligned}
$$

Note from (9.15) that r_{it} holds if and only if there is some sequence of interconnects in the LRS state. This assertion is independent of differing delays in wires or interconnects. That is, wire R_i will eventually have a high voltage with respect to ground as long as the right sequence of interconnects is in the LRS state. It follows that delays arising from process variations or external

factors can be accounted for by increasing the time interval before reading the outputs of the system. Thus, no additional complexity or circuitry need be introduced in order to deal with such timing errors.

All-Pairs Shortest Paths

The all-pairs shortest paths problem seeks to compute the shortest length of a path between any pair of nodes in a network. There is a well-known connection between this problem and the transitive closure operation. In fact, these problems become synonymous when dealing with unweighted graphs. While the shortest paths problem minimizes over additions of edge weights, the transitive closure operation performs logical conjunctions and disjunctions over binary edge weights. In the case of an unweighted graph $G = (V, E)$ with k-reachability matrix $X^{(k)} = (I \vee E)^k$, the shortest path length between two nodes v_α and v_β can be determined using transitive closure by finding the first t for which $X^{(t)}_{\alpha\beta} = 1$, thereby minimizing over the number of edges in a path. Due to these similarities, the mapping proposed in the previous section serves as a precursor to the solution of the single-source and all-pairs shortest paths problems. Such a solution opens up the possibility for a computation-in-memory approach to a variety of interesting problems, such as Gaussian elimination [102], optimal routing [103], and generating regular expressions for Deterministic Finite Automata [104], among others.

Theorem 9 demonstrates that the shortest path length between any two nodes v_α and v_β can be determined using the mapping described in the previous section by monitoring the values r_β, c_β.

Theorem 9. *Let $G = (V, E)$ denote an unweighted graph, where $|V| = n$ and $X^{(1)} = (I \vee E)$ denotes the 1-reachability matrix of G. Given an $n \times n$ crossbar $\mathcal{X} = (M^1, M^2, R, C)$ with $M^1 = X^{(1)}$ and $M^2 = (X^{(1)})^T$, if the shortest path between $v_\alpha, v_\beta \in V$ is of length τ, then $r_{\beta,\tau/2} \wedge \neg c_{\beta,\tau/2}$ holds if τ is even and $\neg r_{\beta,(\tau-1)/2} \wedge c_{\beta,(\tau+1)/2}$ holds if τ is odd.*

Proof. Let $(r_{10}, \ldots, r_{n0}) = (I_{\alpha 0}, \ldots, I_{\alpha n})$. From Theorem 8, we have $r_{it} = ((X^{(1)})^{2t})_{\alpha i}$ and $c_{jt} = ((X^{(1)})^{2t-1})_{\alpha j}$. Two cases arise.

- τ even:

$$r_{\beta,\tau/2} = \left((X^{(1)})^{\tau}\right)_{\alpha\beta} = X^{(\tau)}_{\alpha\beta} \tag{9.16}$$

$$c_{\beta,\tau/2} = \left((X^{(1)})^{\tau-1}\right)_{\alpha\beta} = X^{(\tau-1)}_{\alpha\beta} \tag{9.17}$$

  Since the shortest path is of length τ, it follows that $X^{(\tau-1)}_{\alpha\beta} = 0$ and $X^{(\tau)}_{\alpha\beta} = 1$. Therefore, $r_{\beta,\tau/2} \wedge \neg c_{\beta,\tau/2}$ must hold.

- τ odd:

$$r_{\beta,(\tau-1)/2} = \left((X^{(1)})^{\tau-1}\right)_{\alpha\beta} = X^{(\tau-1)}_{\alpha\beta} \tag{9.18}$$

$$c_{\beta,(\tau+1)/2} = \left((X^{(1)})^{\tau}\right)_{\alpha\beta} = X^{(\tau)}_{\alpha\beta} \tag{9.19}$$

  Since the shortest path is of length τ, it follows that $X^{(\tau-1)}_{\alpha\beta} = 0$ and $X^{(\tau)}_{\alpha\beta} = 1$. Therefore, $\neg r_{\beta,(\tau-1)/2} \wedge c_{\beta,(\tau+1)/2}$ must hold.

□

It follows as a corollary to Theorem 9 that the length of the shortest path from v_α to v_β can be obtained by applying a voltage to wire R_α, grounding wires R_k ($k \neq \alpha$), and reading the values of wires R_β and C_β for every feedback loop iteration until $r_\beta \oplus c_\beta$ holds. This approach allows us to solve the single-source shortest paths problem with some added circuitry required to continuously monitor the values of wires $R_{\beta t}$ and $C_{\beta t}$. Repeating this operation for each $\alpha \in \{1, \ldots, n\}$ yields a solution to the all-pairs shortest paths problem.

Complexity

Given a graph $G = (V, E)$ with $|V| = n$ and a crossbar $\mathcal{X} = (M^1, M^2, R, C)$, the mapping proposed in Theorem 8 uses $\mathcal{O}(n^2)$ devices. Since the maximum length of a path is $n-1$, we need only compute $(I \vee E)^{n-1}$. It follows from Theorem 8 that, if $M^1 = (I \vee E)$ and $M^2 = (I \vee E)^T$, then $c_{j,n/2} = (I \vee E)_{\gamma j}^{2(n/2)-1} = (I \vee E)_{\gamma j}^{n-1}$. Because this computation must be done for every $\gamma \in \{1, \ldots, n\}$, we have a time complexity of $\mathcal{O}(n^2)$ for the transitive closure and all pairs shortest paths problems. The single-source variants of these problems can be solved in $\mathcal{O}(n)$ operations since we only need to compute values for exactly one row vector indexed by $\gamma \in \{1, \ldots, n\}$.

As a point of comparison, a straightforward parallelization of Floyd's algorithm, which is itself a generalization of transitive closure, yields a time complexity of $\mathcal{O}(n^3/p + n^2 \log p)$ given p processors [100]. This leads to a runtime of $\mathcal{O}(n^2 \log n^2)$ using n^2 processors as compared to our $\mathcal{O}(n^2)$ runtime using $\mathcal{O}(n^2)$ devices.

An efficient in-memory hardware solution to the transitive closure problem can be a precursor to other fast graph algorithms. This is due to the dependence of reachability problems for undirected graphs on the transitive closure procedure. This has been well-documented and poses a significant hurdle in the parallelization of several graph algorithms [105] on John von Neumann architectures.

Universal Computation

To the best of our knowledge, the first exposition of transitive closure as a computational model was presented by Skyum and Valiant in [95]. The framework we have proposed in this paper allows for a physical implementation of this theoretical study. However, it is worth noting that much work remains to be done in the pursuit of efficient computation of Boolean formulas through the use of

a transitive closure model.

Consider a Boolean formula $\phi : \{0,1\}^p \mapsto \{0,1\}$ with p variables $b_1, \ldots, b_p$, where a literal is defined as a variable b_i or its negation $\neg b_i$. The formula ϕ is said to be in conjunctive normal form (CNF) if it is structured as a conjunction of disjunctive clauses. That is, $\phi = \bigwedge_i \bigvee_j l_j^i$ is in CNF, where l_j^i is the j^{th} literal in the i^{th} clause. Conversely, the disjunctive normal form (DNF) is structured as a disjunction of conjunctive clauses $\phi = \bigvee_i \bigwedge_j l_j^i$. These forms are universal and thus any model that computes arbitrary CNF or DNF formulas suffices to compute any Boolean formula.

Given a CNF formula ϕ with $|\Phi|$ clauses and k variables per clause, we can construct a graph $G = (V, E)$ with source and terminal vertices $s, t \in V$ such that there is a path from s to t if and only if ϕ evaluates to 1. This can be accomplished in a straightforward manner by designating the presence or absence of an edge to be the value of a literal in ϕ. The details of how we construct G for a given CNF formula can be seen in the following list. See Figure 9.5 for a visualization and Figure 9.6 for an example.

Vertices (CNF)

- Vertex $v_{(i-1)k+j}$ corresponds to the j^{th} literal in the i^{th} clause. Thus, the first $k|\Phi|$ vertices correspond to literals.
- For the i^{th} clause, we create the vertex $v_{k|\Phi|+i}$. Thus, vertices $v_{k|\Phi|+i}, \ldots, v_{(k+1)|\Phi|}$ correspond to clauses. Of these, $v_{(k+1)|\Phi|}$ functions as the terminal node t.
- The source node s is $v_{(k+1)|\Phi|+1}$.

Edges (CNF)

- The edge $(v_{(i-1)k+j}, v_{k|\Phi|+i}) \in E$ connects the j^{th} literal in the i^{th} clause with its corresponding clause vertex. This edge is present if said literal evaluates to 1.
- For $j = 1, \ldots, k$, edges of the form $(v_{(k+1)|\Phi|+1}, v_j) \in E$ connect the source vertex with the literal vertices in the first clause.

We can similarly define a graph construction given a DNF formula. We have chosen to leave the numbering of the vertices corresponding to literals and clauses the same as in the CNF construction for convenience.

Vertices (DNF)

- Vertex $v_{(i-1)k+j}$ corresponds to the j^{th} literal in the i^{th} clause. Thus, the first $k|\Phi|$ vertices correspond to literals.
- For the i^{th} clause, we create the vertex $v_{k|\Phi|+i}$. Thus, vertices $v_{k|\Phi|+i}, \ldots, v_{(k+1)|\Phi|}$ correspond to clauses.
- Vertex $v_{(k+1)|\Phi|+1}$ corresponds to the source node s.
- Vertex $v_{(k+1)|\Phi|+2}$ corresponds to the terminal node t.

Edges (DNF)

- The edge $(v_{(i-1)k+j}, v_{(i-1)k+j+1}) \in E$ is present if the j^{th} literal in the i^{th} clause is 1.
- for all $i = 1, \ldots, |\Phi|$, we have $(v_{(k+1)|\Phi|+1}, v_{(i-1)k+1}) \in E$. These edges correspond to the connections from the source node s to the first literal in each clause.
- for all $i = 1, \ldots, |\Phi|$, we have $(v_{k|\Phi|+i}, v_{(k+1)|\Phi|+2}) \in E$. These edges correspond to connections from each clause to the terminal node t.

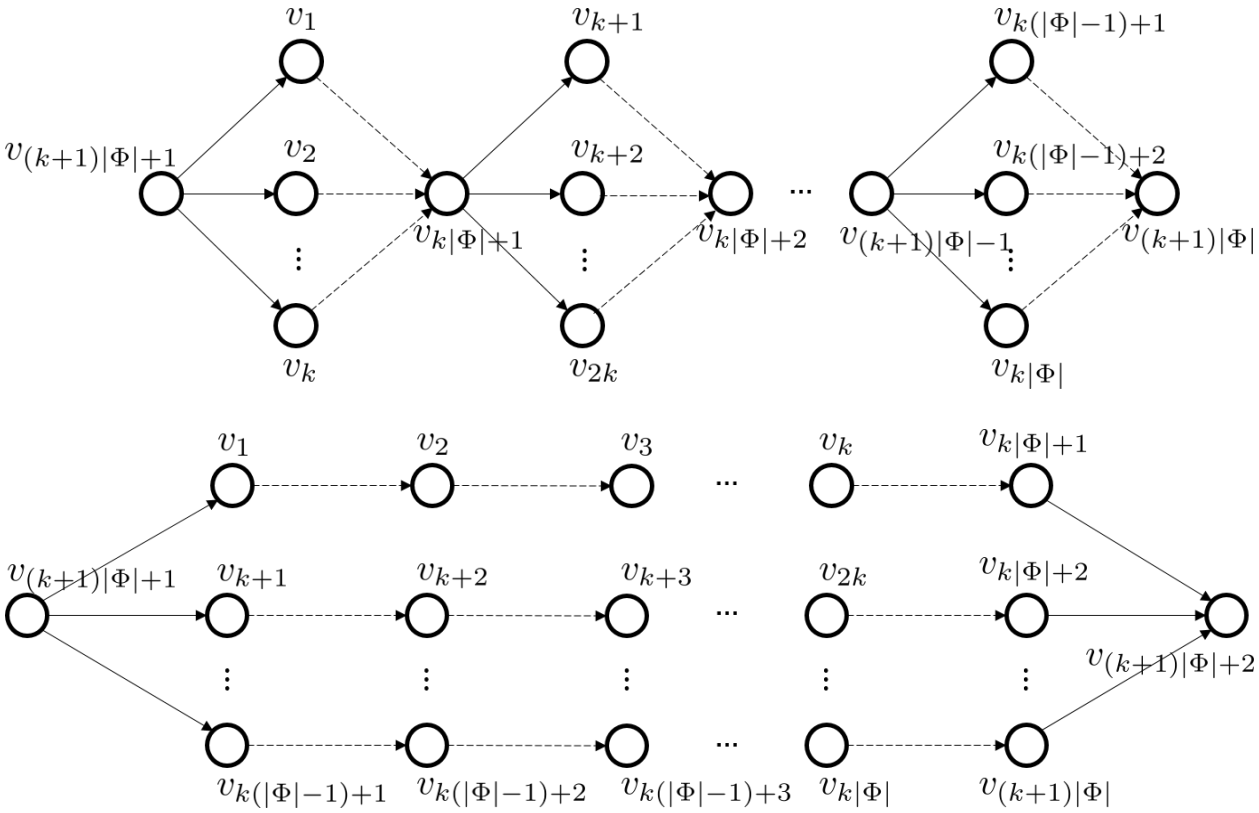


Figure 9.5: Graph constructions for a given CNF (*Top*) and DNF (*Bottom*) formula with $|\Phi|$ clauses and k literals per clause. The dashed lines denote edges whose presence is a function of the literals in the formula as specified in our constructions.

$$X^{\oplus}_{\text{CNF}} = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b2&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b_3&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b_1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_2&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b_3&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b_1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&\neg b_2&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_3&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_2&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&b_3&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix} \tag{9.20}$$

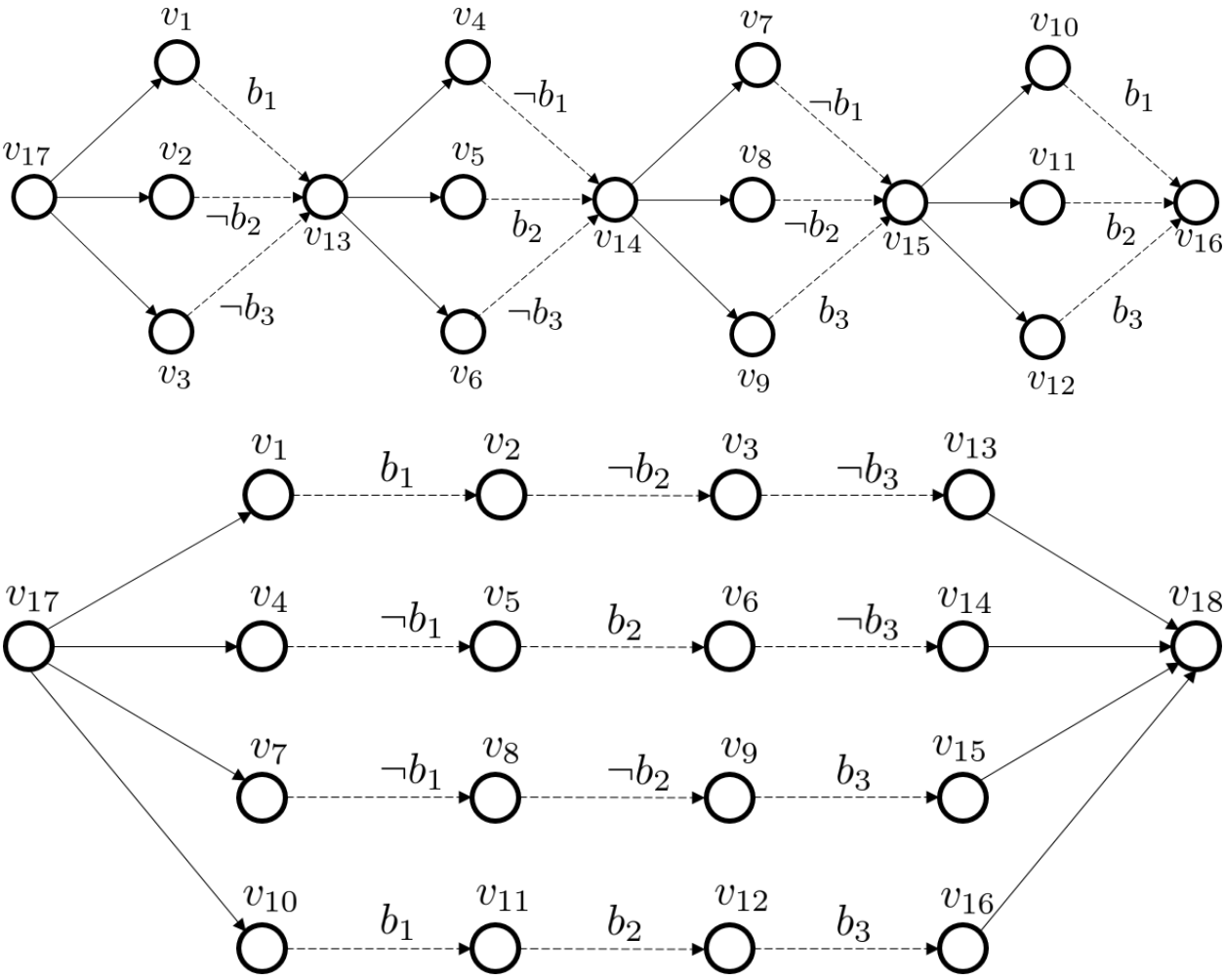


Figure 9.6: (*Top*) Given the CNF 3-bit parity formula $\phi = (b_1 \vee \neg b_2 \vee \neg b_3) \wedge (\neg b_1 \vee b_2 \vee \neg b_3) \wedge (\neg b_1 \vee \neg b_2 \vee b_3) \wedge (b_1 \vee b_2 \vee b_3)$, its corresponding graph construction is given by the adjacency matrix $X^{\oplus}_{\text{CNF}}$ (9.20). (*Bottom*) Given the DNF 3-bit parity formula $\phi = (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge b_3) \vee (b_1 \wedge b_2 \wedge b_3)$, its corresponding graph construction is given by the adjacency matrix $X^{\oplus}_{\text{DNF}}$ (9.21).

$$X^{\oplus}_{\text{DNF}} = \begin{pmatrix}
0 & b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \neg b_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \neg b_3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \neg b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & b_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \neg b_3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \neg b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \neg b_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \tag{9.21}$$

Suppose we are given a 3D crossbar $\mathbb{C} = (M^1, M^2, R, C)$ and a CNF formula ϕ_{CNF}. It follows from Theorem 8 that, if we set $M^1 = X^{\oplus}_{\text{CNF}}$, $M^2 = (X^{\oplus}_{\text{CNF}})^T$, and $\gamma = (k+1)|\Phi| + 1$, then we will read the wire value $r_{(k+1)|\Phi|} = \phi_{\text{CNF}}$. Similarly, given a DNF formula ϕ_{DNF}, we can set $M^1 = X^{\oplus}_{\text{DNF}}$, $M^2 = (X^{\oplus}_{\text{DNF}})^T$, and $\gamma = (k+1)|\Phi| + 1$ to read the wire value $r_{(k+1)|\Phi|+2} = \phi_{\text{DNF}}$.

While the constructions presented in this section function as an interesting proof of concept, they require crossbars of dimension $\mathcal{O}(k|\Phi|) \times \mathcal{O}(k|\Phi|)$. As such, the authors believe that this remains a theoretical curiosity that is not yet practical enough to be implemented in real systems. However, it is a first step in the direction of a universal in-memory computation model based on transitive closure.

Experimental Results

In order to cement the validity and effectiveness of our approach, simulations were carried out using HSPICE and NVSim. The HSPICE simulations were performed under the following parameters. The resistive components of the 1D1R interconnects corresponding to $m^k_{ij} = 1$ have a resistance of 1 mΩ, devices corresponding to $m^k_{ij} = 0$ have a resistance of 1 GΩ, and resistors to ground have a resistance of 1 kΩ. For our choice of diode model, we decided to use the Super Barrier Rectifier (SBR) SPICE model SBR3U20SA designed by Diodes Incorporated® [85].

We test our approach on random Boolean matrices of various sizes. For crossbars of sizes 4×4, 8×8, 16×16, 32×32, 64×64, and 128×128, we sample 100 random matrices and compute their transitive closure using our approach. See Table 9.2 for results. It can be seen that there is a margin of two orders of magnitude between 1 and 0 values in the average case and there is an order of magnitude gap in the worst case; that is, in the case where we compare the minimal 1 value and the maximal 0 value.

In defining these random matrices, we utilize a Bernoulli distribution to determine the probability $P(v_i, v_j)$ of two nodes being connected by an edge. We set this probability as the threshold function (9.22) for Bernoulli graphs [106].

$$T(n) = \frac{\log(n)}{n} \tag{9.22}$$

Given a graph $G = (V, E)$, we choose connectivity probabilities $P(v_i, v_j) = T(|V|)$ according to equation (9.22) for the simulations in Table 9.2. We choose this threshold function because it can be shown that if $P(v_i, v_j) = kT(|V|)$, the probability of a graph being connected approaches 0 (1) if $k < 1$ ($k > 1$). This provides a good mix of values in our transitive closure matrices.

Energy and Latency Considerations

In order to test the latency and energy efficiency of our approach, we model our architecture within the NVSim non-volatile memory simulation environment [7]. For the resistive component of the 1D1R interconnects, we modeled the memristor studied in [86]. This memristor exhibits a switching time of 85 ps under a 2 V pulse with switching currents below 15 μA, leading to a switching energy of 3 fJ. The read and write latencies and energies for memories of various sizes under this

Table 9.2: HSPICE simulation results of graph transitive closure computation using our 3D crossbars. The first column denotes crossbar sizes. For each row in the table, the transitive closure of 100 random matrices were computed and the minimal voltage reading corresponding to a value of 1, or *true*, was recorded in the first column. The maximal value corresponding to a value of 0 was also recorded, as were the average and standard deviations of *false* and *true* voltage readings. It can be seen that there is a two-orders-of-magnitude gap between *true* and *false* readings in the average case and there is an order of magnitude gap in the worst case; that is, in the case where we compare the minimal *true* value and the maximal *false* value. Circuit generation files can be found in eecs.ucf.edu/∼velasquez/TransitiveClosure.

Crossbar dimensions	Minimal *true* reading	Maximal *false* reading	Average *true* reading	Average *false* reading	Standard Deviation of *true* readings	Standard Deviation of *false* readings
4×4	4.3156 V	0.0480 V	4.7615 V	0.0129 V	0.5049 V	0.0012 V
8×8	3.9149 V	0.0800 V	4.6341 V	0.0194 V	0.9855 V	0.0027 V
16×16	3.8664 V	0.0960 V	4.5495 V	0.0136 V	1.4698 V	0.0051 V
32×32	3.8324 V	0.1280 V	4.4959 V	0.0326 V	1.6871 V	0.0131 V
64×64	3.8383 V	0.1601 V	4.4231 V	0.1021 V	2.2237 V	0.0050 V
128×128	3.8303 V	0.1762 V	4.3458 V	0.1195 V	2.7045 V	0.0061 V

configuration are presented in Table 9.3. We use these values to determine the computation time and energy consumed by our approach when computing the transitive closure of graphs with tens of thousands of nodes. These results are reported in Table 9.4. It can be seen that the transitive closure of networks with tens of thousands of nodes can be computed using less energy (1.225 nJ for the ca-HepPh benchmark) than it takes to access DRAM ($1.3 - 2.6$ nJ) [24].

Note that the write latency and write energy dominate the computation time and energy consumed in the benchmarks presented. Thus, it is important to further develop the graph-theoretic platform presented in this paper so that multiple algorithms may be run on the stored graph data. This would allow us to only configure the memory once, thereby minimizing latency and energy usage.

Table 9.3: Latency and power metrics of the proposed architecture for various memory capacities using the NVSim [7] simulator. NVSIM files can be found in eecs.ucf.edu/∼velasquez/TransitiveClosure.

Memory capacity	Read latency (ns)	Write latency (ns)	Read energy (nJ)	Write energy (nJ)
1 MB	1.535	1.292	0.016	0.013
4 MB	2.200	2.100	0.029	0.25
16 MB	4.131	4.413	0.064	0.049
64 MB	10.028	11.459	0.175	0.098
256 MB	30.661	35.925	0.768	0.194

Table 9.4: Computation time and energy usage metrics for various benchmark circuits taken from the Stanford Large Network Data Collection. These networks are mapped to the memories presented in Table 9.3 and the aforementioned metrics are reported. In the (Read) columns, we report values that correspond to memories whose cells contain the contents of the network. The (Write + Read) columns assume that the memories must first be configured to contain the contents of the network.

Benchmark	Number of Nodes	Time (Read)	Time (Write + Read)	Energy (Read)	Energy (Write + Read)
ego-Facebook	4,039	2.712 ns	17.206 μs	0.116 nJ	0.205 μJ
wiki-Vote	7,115	5.923 ns	72.309 μs	0.256 nJ	0.803 μJ
p2p-Gnutella09	8,114	6.691 ns	72.309 μs	0.320 nJ	0.803 μJ
ca-HepPh	12,008	23.34 ns	0.375 ms	1.225 nJ	3.208 μJ
ca-AstroPh	18,772	87.977 ns	2.351 ms	5.376 nJ	12.699 μJ
p2p-Gnutella25	22,687	71.601 ns	2.352 ms	3.840 nJ	12.699 μJ
ca-CondMat	23,133	87.977 ns	2.352 ms	5.376 nJ	12.699 μJ
p2p-Gnutella24	26,518	71.601 ns	2.352 ms	3.840 nJ	12.699 μJ
cit-HepTh	27,770	83.883 ns	2.352 ms	5.376 nJ	12.699 μJ

Concluding Remarks

A new in-memory computing design for obtaining the transitive closure of a graph using a crossbar with two layers of interconnects has been presented in this chapter. This procedure only requires the addition of external feedback loops to already existing 3D memories. We have also shown that

the approach is both fast and energy-efficient on practical networks obtained from the Stanford Large Network Data Collection.

We plan to extend this framework to solve the problem of parsing context-free grammars (CFG) efficiently within memory as it has been shown that fast CFG parsing requires efficient transitive closure algorithms [30]. We also plan to explore the hardware parallelization of other graph algorithms via in-memory computing as we believe that success in this area would bring us much closer to a unified in-memory computing architecture.

CHAPTER 10: CONCLUSION

In this thesis, we have demonstrated how computation can be performed within crossbar memories by leveraging the paths of current between two wires. The theory of paths-based logic was developed as a foundation to argue mathematically about the meaning of paths when interconnects in the memory are programmed according to variables in a given function. This theory yielded some surprising results. Namely, that a small subset of paths between two wires has the same computing power as all of the paths in the crossbar. This result was leveraged in our bounded model checking procedures by providing an upper bound on the number transitions that the model checking automata has to make in order to determine whether a design is valid for a given formula. A robust fault tolerance model was also proposed and verified experimentally. Since the complexity of model checking procedures is exponential in the number of parameters, the proposed framework was restricted to generating designs for small functions. We argued that such a constraint can be overcome by separating a function into sub-formula and using a network of inter-connected crossbars to compute the function by using the output of one crossbar as input to the next. However, we proved that in this case, paths-based logic on homogeneous crossbars is no longer universal and a more expressive crossbar model is required. We demonstrated an effective solution by introducing heterogeneous crossbars and established how our approach leads to state-of-the-art fault-tolerant addition circuits in these non-traditional crossbars. An alternative synthesis technique that produces designs which are state-of-the-art in terms of space was also presented. This technique searches through the space of valid voltage sequences as opposed to memory configurations.

In the later chapters, we looked at two problems of interest. First, the problem of computing the product of a set of Boolean matrices was investigated. A 3-dimensional crossbar solution that is parsimonious in space and execution time was presented. The second problem we considered was that of computing the transitive closure of a network using only two layers of interconnects. We

proposed a solution and rigorously proved its correctness as well as verified it experimentally on randomized matrices and real-world networks with tens of thousands of nodes.

LIST OF REFERENCES

[1] Tezaswi Raja and Samiha Mourad. Digital logic implementation in memristor-based crossbars-a tutorial. In *Electronic Design, Test and Application, 2010. DELTA'10. Fifth IEEE International Symposium on*, pages 303–309. IEEE, 2010.

[2] Eero Lehtonen, Jussi H Poikonen, and Mika Laiho. Applications and limitations of memristive implication logic. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–6. IEEE, 2012.

[3] Alvaro Velasquez and Sumit Kumar Jha. Parallel computing using memristive crossbar networks: Nullifying the processor-memory bottleneck. In *Design & Test Symposium (IDT), 2014 9th International*, pages 147–152. IEEE, 2014.

[4] Luca Pilato, Sergio Saponara, and Luca Fanucci. Performance of digital adder architectures in 180nm cmos standard-cell technology. In *Applied Electronics (AE), 2016 International Conference on*, pages 211–214. IEEE, 2016.

[5] Matt Poremba, Sparsh Mittal, Dong Li, Jeffrey S Vetter, and Yuan Xie. Destiny: A tool for modeling emerging 3d nvm and edram caches. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1543–1546. EDA Consortium, 2015.

[6] Béla Bollobás and Oliver M Riordan. Mathematical results on scale-free random graphs. *Handbook of graphs and networks: from the genome to the internet*, pages 1–34, 2003.

[7] Xiangyu Dong, Cong Xu, Norm Jouppi, and Yuan Xie. Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*, pages 15–50. Springer, 2014.

[8] Andre DeHon and Benjamin Gojman. Crystals and snowflakes: building computation from nanowire crossbars. *Computer*, 44(2):0037–45, 2011.

[9] Dmitri B Strukov and Konstantin K Likharev. Reconfigurable nano-crossbar architectures. *Nanoelectronics and Information Technology*, pages 543–562, 2012.

[10] Mark Neisser and Stefan Wurm. Itrs lithography roadmap: 2015 challenges. *Advanced Optical Technologies*, 4(4):235–240, 2015.

[11] Seong Jun Kang, Coskun Kocabas, Taner Ozel, Moonsub Shim, Ninad Pimparkar, Muhammad A Alam, Slava V Rotkin, and John A Rogers. High-performance electronics using dense, perfectly aligned arrays of single-walled carbon nanotubes. *Nature nanotechnology*, 2(4):230–236, 2007.

[12] Tezaswi Raja and Samiha Mourad. Digital logic implementation in memristor-based crossbars-a tutorial. In *Electronic Design, Test and Application, 2010. DELTA'10. Fifth IEEE International Symposium on*, pages 303–309. IEEE, 2010.

[13] Bao Liu. Advancements on crossbar-based nanoscale reconfigurable computing platforms. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 17–20. IEEE, 2010.

[14] Yehua Su and Wenjing Rao. Defect-tolerant logic hardening for crossbar-based nanosystems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 1801–1806. IEEE, 2013.

[15] Yehua Su and Wenjing Rao. An integrated framework toward defect-tolerant logic implementation onto nanocrossbars. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 33(1):64–75, 2014.

[16] Bo Yuan, Xin Yao, Bin Li, and W Thomas. A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nano-architectures. 2014.

[17] Masoud Zamani and Mehdi Baradaran Tahoori. Variation-aware logic mapping for crossbar nano-architectures. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 317–322. IEEE, 2011.

[18] Kimberly Keeton. Memory-driven computing. In *FAST*, 2017.

[19] Intel PR. Intel and micron produce breakthrough memory technology. Accessed: 28 July, 2015.

[20] Katherine Bourzac. Has intel created a universal memory technology?[news]. *IEEE Spectrum*, 54(5):9–10, 2017.

[21] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3d xpoint technology. *Proceedings of the IEEE*, 2017.

[22] Sandisk PR. Sandisk and hp launch partnership to create memory-driven computing solutions. `https://www.sandisk.com/about/media-center/press-releases/2015/sandisk-and-hp-launch-partnership`. Accessed: 8 October, 2015.

[23] Andreas Nowatzyk, Fong Pong, and Ashley Saulsbury. Missing the memory wall: The case for processor/memory integration. In *Computer Architecture, 1996 23rd Annual International Symposium on*, pages 90–90. IEEE, 1996.

[24] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 10–14. IEEE, 2014.

[25] Bill Dally. Challenges for future computing systems. https://www.hipeac.net/events/activities/7151/challenges-for-future-computing-systems/, January 19 2015. HiPEAC 2015 Keynote speech.

[26] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[27] George K Atia and Venkatesh Saligrama. Boolean compressed sensing and noisy group testing. *Information Theory, IEEE Transactions on*, 58(3):1880–1901, 2012.

[28] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 297–306. IEEE, 2008.

[29] Yeghisabet Alaverdyan and Gevorg Margarov. Fast asymmetric cryptosystem based on boolean product of matrices. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 392–395. IEEE, 2009.

[30] Leslie G Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.

[31] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM (JACM)*, 49(1):1–15, 2002.

[32] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[33] Jie Sun, Erik Lind, Ivan Maximov, and HQ Xu. Memristive and memcapacitive characteristics of a au/ti—inp/ingaas diode. *Electron Device Letters, IEEE*, 32(2):131–133, 2011.

[34] Sung Hyun Jo and Wei Lu. Cmos compatible nanoscale nonvolatile resistance switching memory. *Nano Letters*, 8(2):392–397, 2008.

[35] John Paul Strachan, Gilberto Medeiros-Ribeiro, J Joshua Yang, M-X Zhang, Feng Miao, Ilan Goldfarb, Martin Holt, Volker Rose, and R Stanley Williams. Spectromicroscopy of tantalum oxide memristors. *Applied Physics Letters*, 98(24):242114, 2011.

[36] Dongqing Liu, Chaoyang Zhang, Guang Wang, Zhengzheng Shao, Xuan Zhu, Nannan Wang, and Haifeng Cheng. Nanoscale electrochemical metallization memories based on amorphous (la, sr) mno3 using ultrathin porous alumina masks. *Journal of Physics D: Applied Physics*, 47(8):085108, 2014.

[37] Hussein Nili, Sumeet Walia, Sivacarendran Balendhran, Dmitri B Strukov, Madhu Bhaskaran, and Sharath Sriram. Nanoscale resistive switching in amorphous perovskite oxide (a-srtio3) memristors. *Advanced Functional Materials*, 24(43):6741–6750, 2014.

[38] Kuk-Hwan Kim, Sung Hyun Jo, Siddharth Gaba, and Wei Lu. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. *Applied Physics Letters*, 96(5):053106, 2010.

[39] Seth Copen Goldstein and Mihai Budiu. Nanofabrics: Spatial computing using molecular electronics. *ACM SIGARCH Computer Architecture News*, 29(2):178–191, 2001.

[40] Andre DeHon and Michael J Wilson. Nanowire-based sublithographic programmable logic arrays. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 123–132. ACM, 2004.

[41] Teng Wang, Zhenghua Qi, and Csaba Andras Moritz. Opportunities and challenges in application-tuned circuits and architectures based on nanodevices. In *Proceedings of the 1st conference on Computing frontiers*, pages 503–511. ACM, 2004.

[42] Pavan Panchapakeshan, Pritish Narayanan, and Csaba Andras Moritz. N3asics: designing nanofabrics with fine-grained cmos integration. In *Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium on*, pages 196–202. IEEE, 2011.

[43] Greg Snider, Philip Kuekes, and R Stanley Williams. Cmos-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881, 2004.

[44] Xialong Ma, Dmitri B Strukov, Jung Hoon Lee, and Konstantin K Likharev. Afterlife for silicon: Cmol circuit architectures. In *Nanotechnology, 2005. 5th IEEE Conference on*, pages 175–178. IEEE, 2005.

[45] Gregory S Snider and R Stanley Williams. Nano/cmos architectures using a field-programmable nanowire interconnect. *Nanotechnology*, 18(3):035204, 2007.

[46] D Tu, M Liu, Wei Wang, and S Haruehanroengra. Three-dimensional cmol: Three-dimensional integration of cmos/nanomaterial hybrid digital circuits. *Micro & Nano Letters, IET*, 2(2):40–45, 2007.

[47] Wei Fei, Hao Yu, Wei Zhang, and Kiat Seng Yeo. Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(6):1012–1025, 2012.

[48] Jun Wu and Minsu Choi. Memristor lookup table (mlut)-based asynchronous nanowire crossbar architecture. In *Nanotechnology (IEEE-NANO), 2010 10th IEEE Conference on*, pages 1100–1103. IEEE, 2010.

[49] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. Memristor-based memory: the sneak paths problem and solutions. *Microelectronics Journal*, 44(2):176–183, 2013.

[50] Kuk-Hwan Kim, Sung Hyun Jo, Siddharth Gaba, and Wei Lu. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. *Applied Physics Letters*, 96(5):053106, 2010.

[51] Alvaro Velasquez and Sumit Kumar Jha. Automated synthesis of crossbars for nanoscale computing using formal methods. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 130–136. IEEE, 2015.

[52] Mehdi B Tahoori. Application-independent defect tolerance of reconfigurable nanoarchitectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(3):197–218, 2006.

[53] Bo Yuan and Bin Li. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(3):25, 2014.

[54] André DeHon. Array-based architecture for fet-based, nanoscale electronics. *Nanotechnology, IEEE Transactions on*, 2(1):23–32, 2003.

[55] Y. Cassuto, S. Kvatinsky, and E. Yaakobi. Sneak-path constraints in memristor crossbar arrays. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 156–160, 2013.

[56] M Zidan, A Eltawil, Fadi Kurdahi, H Fahmy, and K Salama. Memristor multi-port readout: A closed-form solution for sneak-paths. 2014.

[57] Rawan Naous, Mohammed Affan Zidan, Ahmad Sultan-Salem, and Khaled Nabil Salama. Memristor based crossbar memory array sneak path estimation. In *Cellular Nanoscale Networks and their Applications (CNNA), 2014 14th International Workshop on*, pages 1–2. IEEE, 2014.

[58] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. Memristor-based memory: the sneak paths problem and solutions. *Microelectronics Journal*, 44(2):176–183, 2013.

[59] Pascal O Vontobel, Warren Robinett, Philip J Kuekes, Duncan R Stewart, Joseph Straznicky, and R Stanley Williams. Writing to and reading from a nano-scale crossbar memory based on memristors. *Nanotechnology*, 20(42):425204, 2009.

[60] Harika Manem, Garrett S Rose, Xiaoli He, and Wei Wang. Design considerations for variation tolerant multilevel cmos/nano memristor memory. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 287–292. ACM, 2010.

[61] Seungjun Kim, Hu Young Jeong, Sung Kyu Kim, Sung-Yool Choi, and Keon Jae Lee. Flexible memristive memory array on plastic substrates. *Nano letters*, 11(12):5438–5442, 2011.

[62] Chul-Moon Jung, Jun-Myung Choi, and Kyeong-Sik Min. Two-step write scheme for reducing sneak-path leakage in complementary memristor array. *Nanotechnology, IEEE Transactions on*, 11(3):611–618, 2012.

[63] J Joshua Yang, M-X Zhang, Matthew D Pickett, Feng Miao, John Paul Strachan, Wen-Di Li, Wei Yi, Douglas AA Ohlberg, Byung Joon Choi, Wei Wu, et al. Engineering nonlinearity into memristors for passive crossbar applications. *Applied Physics Letters*, 100(11):113501, 2012.

[64] MS Qureshi, W Yi, G Medeiros-Ribeiro, and RS Williams. Ac sense technique for memristor crossbar. *Electronics letters*, 48(13):757–758, 2012.

[65] Ella Gale, Ben de Lacy Costello, and Andrew Adamatzky. *Boolean Logic Gates from a Single Memristor via Low-Level Sequential Logic*, pages 79–89. Springer, 2013.

[66] Shahar Kvatinsky, Avinoam Kolodny, Uri C Weiser, and Eby G Friedman. Memristor-based imply logic design procedure. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 142–147. IEEE.

[67] Eero Lehtonen and Mika Laiho. Stateful implication logic with memristors. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 33–36. IEEE Computer Society, 2009.

[68] Eero Lehtonen, Jussi Poikonen, and Mika Laiho. Implication logic synthesis methods for memristors. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2441–2444. IEEE, 2012.

[69] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Memristor-based material implication (imply) logic: Design principles and methodologies. 2013.

[70] P Kuekes. Material implication: digital logic with memristors. In *Memristor and memristive systems symposium*, volume 21, 2008.

[71] Divya Mahajan, Matheen Musaddiq, and Earl E Swartzlander. Memristor based adders. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, pages 1256–1260. IEEE, 2014.

[72] Yuanfan Yang, Jimson Mathew, Salvatore Pontarelli, Marco Ottavi, and Dhiraj K Pradhan. Complementary resistive switch-based arithmetic logic implementations using material implication. *IEEE Transactions on Nanotechnology*, 15(1):94–108, 2016.

[73] Anne Siemon, Stephan Menzel, Rainer Waser, and Eike Linn. A complementary resistive switch-based crossbar array adder. *IEEE journal on emerging and selected topics in circuits and systems*, 5(1):64–74, 2015.

[74] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magicmemristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[75] Eero Lehtonen, Jussi Poikonen, and Mika Laiho. Implication logic synthesis methods for memristors. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2441–2444. IEEE, 2012.

[76] Sumit Jha, Dilia Rodriguez, Joseph E. Van Nostrand, and Alvaro Velasquez. Computation of boolean formulas using sneak paths in crossbar computing, 2014. Work-in-Progress Poster presented at the Design Automation Conference, June 1 – 5, San Francisco, CA.

[77] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 19(1):7–34, 2001.

[78] Alvaro Velasquez and Sumit Kumar Jha. Fault-tolerant in-memory crossbar computing using quantified constraint solving. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 101–108. IEEE, 2015.

[79] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha. Flow-based computing on nanoscale crossbars: Design and implementation of full adders. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1870–1873, May 2016.

[80] Sezer Gören, H Fatih Ugurdag, and Okan Palaz. Defect-aware nanocrossbar logic mapping through matrix canonization using two-dimensional radix sort. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 7(3):12, 2011.

[81] Yehua Su and Wenjing Rao. Defect-tolerant logic mapping on nanoscale crossbar architectures and yield analysis. In *Defect and Fault Tolerance in VLSI Systems, 2009. DFT'09. 24th IEEE International Symposium on*, pages 322–330. IEEE, 2009.

[82] Dilia E Rodriguez, Joseph E Van Nostrand, Sumit Jha, and Alvaro Velasquez. Computation of boolean formulas using sneak paths in crossbar computing, December 17 2014. US Patent App. 14/573,677.

[83] Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev. Nusmv 2.5 user manual, 2010. *URL http://nusmv. fbk. eu/NuSMV/userman/v25/nusmv. pdf. Accessed on June*, 24, 2013.

[84] A. Velasquez and S. K. Jha. Parallel boolean matrix multiplication in linear time using rectifying memristors. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1874–1877, May 2016.

[85] Diodes Incorporated. Spice models. http://www.diodes.com/spicemodels/search.php. Accessed: 2016-12-04.

[86] Byung Joon Choi, Antonio C Torrezan, John Paul Strachan, PG Kotula, AJ Lohn, Matthew J Marinella, Zhiyong Li, R Stanley Williams, and J Joshua Yang. High-speed and low-energy nitride memristors. *Advanced Functional Materials*, 2016.

[87] Jens Bürger, Christof Teuscher, and Marek Perkowski. Digital logic synthesis for memristors. *Reed-Muller 2013*, pages 31–40, 2013.

[88] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. Logic synthesis for majority based in-memory computing. In *Advances in Memristors, Memristive Devices and Systems*, pages 425–448. Springer, 2017.

[89] Eero Lehtonen, JH Poikonen, and Mika Laiho. Two memristors suffice to compute all boolean functions. *Electronics letters*, 46(3):239–240, 2010.

[90] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. Metal–oxide rram. *Proceedings of the IEEE*, 100(6):1951–1970, 2012.

[91] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. Design of cross-point metal-oxide reram emphasizing reliability and cost. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 17–23. IEEE, 2013.

[92] Eero Lehtonen, Jussi H Poikonen, and Mika Laiho. Applications and limitations of memristive implication logic. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–6. IEEE, 2012.

[93] Mika Laiho and Eero Lehtonen. Arithmetic operations within memristor-based analog memory. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–4. IEEE, 2010.

[94] Lynn E Cannon. A cellular computer to implement the kalman filter algorithm. Technical report, DTIC Document, 1969.

[95] Sven Skyum and Leslie G Valiant. A complexity theory based on boolean algebra. *Journal of the ACM (JACM)*, 32(2):484–502, 1985.

[96] Juan L Reutter, Miguel Romero, and Moshe Y Vardi. Regular queries on graph databases. In *18th International Conference on Database Theory (ICDT 2015)*, volume 31, pages 177–194. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[97] Dick Grune, Kees Van Reeuwijk, Henri E Bal, Ceriel JH Jacobs, and Koen Langendoen. *Modern compiler design*. Springer Science & Business Media, 2012.

[98] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418. Springer, 2000.

[99] E Nuutila. Efficient transitive closure computation in large digraphs, mathematics and computing in engineering series no. 74 phd thesis helsinki university oftechnology, 1995.

[100] Michael J Quinn. *Parallel Programming*, volume 526. TMH CSE, 2003.

[101] Marc D Riedel and Jehoshua Bruck. Cyclic boolean circuits. *Discrete Applied Mathematics*, 160(13):1877–1900, 2012.

[102] Rezaul Alam Chowdhury and Vijaya Ramachandran. The cache-oblivious gaussian elimination paradigm: theoretical framework and experimental evaluation. In *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 236–236. ACM, 2006.

[103] Alvaro Velasquez, Piotr Wojciechowski, K Subramani, Steven L Drager, and Sumit Kumar Jha. The cardinality-constrained paths problem: Multicast data routing in heterogeneous communication networks. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 126–130. IEEE, 2016.

[104] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Technical report, DTIC Document, 1951.

[105] Ming-Yang Kao and Philip N Klein. Towards overcoming the transitive-closure bottleneck: Efficient parallel algorithms for planar digraphs. *Journal of Computer and System Sciences*, 47(3):459–500, 1993.

[106] Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.