2019

Automated Synthesis of Memristor Crossbar Networks

Dwaipayan Chakraborty
University of Central Florida

AUTOMATED SYNTHESIS OF MEMRISTOR CROSSBAR NETWORKS

by

DWAIPAYAN CHAKRABORTY
B.Tech, Maulana Abul Kalam Azad University of Technology, West Bengal, 2015

A dissertation submitted in partial fulfilment 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

Summer Term
2019

Major Professor: Sumit Kumar Jha

ABSTRACT

The advancement of semiconductor device technology over the past decades has enabled the design of increasingly complex electrical and computational machines. Electronic design automation (EDA) has played a significant role in the design and implementation of transistor-based machines. However, as transistors move closer toward their physical limits, the speed-up provided by Moore's law will grind to a halt. Once again, we find ourselves on the verge of a paradigm shift in the computational sciences as newer devices pave the way for novel approaches to computing. One of such devices is the memristor – a resistor with non-volatile memory.

Memristors can be used as junctional switches in crossbar circuits, which comprise of intersecting sets of vertical and horizontal nanowires. The major contribution of this dissertation lies in automating the design of such crossbar circuits – doing a new kind of EDA for a new kind of computational machinery. In general, this dissertation attempts to answer the following questions:

a. How can we synthesize crossbars for computing large Boolean formulas, up to 128-bit?

b. How can we synthesize more compact crossbars for small Boolean formulas, up to 8-bit?

c. For a given loop-free C program doing integer arithmetic, is it possible to synthesize an equivalent crossbar circuit?

We have presented novel solutions to each of the above problems. Our new, proposed solutions resolve a number of significant bottlenecks in existing research, via the usage of innovative logic representation and artificial intelligence techniques. For large Boolean formulas (up to 128-bit), we have utilized Reduced Ordered Binary Decision Diagrams (ROBDDs) to automatically synthesize linearly growing crossbar circuits that compute them. This cutting edge approach towards flow-based computing has yielded state-of-the-art results. It is worth noting that this approach is scalable to n-bit Boolean formulas. We have made significant original contributions by leveraging

artificial intelligence for automatic synthesis of compact crossbar circuits. This inventive method has been expanded to encompass crossbar networks with 1D1M (1-diode-1-memristor) switches, as well. The resultant circuits satisfy the tight constraints of the Feynman Grand Prize challenge and are able to perform 8-bit binary addition. A leading edge development for end-to-end computation with flow-based crossbars has been implemented, which involves methodical translation of loop-free C programs into crossbar circuits via automated synthesis. The original contributions described in this dissertation reflect the substantial progress we have made in the area of electronic design automation for synthesis of memristor crossbar networks.

ACKNOWLEDGMENTS

First and foremost, I want to thank University of Central Florida and the Department of EECS for giving me the opportunity to pursue a PhD – it has been a transformative experience to say the least. I have endless gratitude towards my advisor, Dr. Sumit Jha, for a large number of things. From him, I have tried to learn how to do "good science", the nuances of doing academic research, the necessary skills to survive in the academic environment and a great many things about life in general. Most of all, I am thankful for his never-ending patience during the times when I might not have been an ideal student. The experiences that he has shared and the conversations we have had, have shaped me as a person at a very fundamental level. After four long years of graduate school, the lessons that I have learned from Dr. Jha are also the ones that I will cherish for the rest of my life.

I am very thankful to Dr. Gary Leavens for providing extremely useful insights on my research and dissertation. I am fortunate to have Dr. Sharma Thankachan, Dr. Rickard Ewetz and Dr. Mengyu Xu on my doctoral dissertation advisory committee – I really appreciate the time and effort that they have invested towards guiding me through this process. I would like to thank all committee members, Dr. Xu, Dr. Ewetz, Dr. Thankachan, Dr. Leavens and Dr. Jha for carefully reading my dissertation and providing useful suggestions.

My doctoral research was supported by the National Science Foundation's projects for Software and Hardware Foundations(award #1438989) and Exploiting Parallelism and Scalability (award #1422257). I would like to acknowledge the support provided through AFOSR Young Investigator Award to Dr. Sumit Jha and the Air Force Office of Scientific Research projects (award number FA9550-16-1-0255). I would like to thank UCF for the financial support provided through Graduate Teaching Assistantship opportunities during Summer 2016, Summer 2017, Fall 2017 and Spring 2018. I would also like to thank Jeanine Clements and Olga Rivera for their help and support.

During the course of my graduate studies, I have had the good fortune of collaborating with Sunny

Raj, Amad Ul Hassen, Julio Gutierrez, Troyle Thomas and Dr. Steven Fernandes. They have some of the brightest minds I have ever encountered, and it has been a joyful experience working with them.

Obtaining a doctorate is undeniably a stressful endeavor, especially when it is put in the context of one's life experiences. As such, it demands that the pursuer is able to wield their mental capabilities to the fullest extent. I want to thank Kristal Pollack and Robert Dwyer for their enormous contribution to this end – they have guided me know myself better and enabled me to effectively deal with my internal turmoil. I would not have survived this phase of my life without the unconditional love of my canine companions, Ringo and Brussels – it has helped me push through the darkest of times and their contribution to my life is unquantifiable. Happiness is real only if it is shared, and I am ever so grateful to have spent time with my friends Vikram, Shane, Maya, Jeanice, Lietsel, Amirfarhad, Freddy, Emily, Alma and Lauren. It's been real.

Lastly, I would like to thank my parents for everything. The sacrifices that they have made over the course of their entire lives have enabled me to reach for my wildest dreams and pursue a doctoral degree in a foreign land filled with uncertainties. I am forever indebted to them for weathering the endless emotional turbulence that I have put them through. This thesis is dedicated to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Moore's Law and its limitations

Over the last five decades, Moore's Law [1, 2], has been a driving factor for immense technological and societal progress. While the advancements have indeed validated the status of Moore's observations and projections as a "law" they certainly did not start out as such. The seminal paper simply explores the trend of fabrication parameters – namely approximate component count, die area and device density - and breaks down the complexity of chip fabrication into its contributing factors. These factors are a function of the ones that had previously been observed - die size, dimension reduction, the product of the above two and a generous smattering of device and circuit design cleverness. Moore then proceeds to project the complexity curve for the future, which has shown to be the most impactful contribution of the paper so far, since it predicts the increase in component count per chip with time. Akin to how Moore's Law paints a picture for the future of chip design, Dennard scaling [3] examines possibilities for the scaling of devices - in this case, MOSFETs. The two-dimensional effects of the semiconductor material are kept at a sub-perceptual level, and the authors proceed to develop a design methodology to manufacture devices that have a marked performance improvement over contemporary devices. Further, physical limitations for MOSFET fabrication are investigated and the authors identify a set of problems that need to be solved in order to achieve better performance. We can thus see how a potent combination of chip design methodology and device fabrication technology had driven the industry to its current heights. With the passage of time and advent of technology, Moore's Law was shown to be less of a prediction and more of an oracle guiding the innovation of computational machines [4]. It has held true throughout the MSI (medium scale integration), LSI (large scale integration) and VLSI (very large scale integration) stages of circuit fabrication.

As had been projected by Moore's Law, transistor scaling has kept on improving. However, Dennard scaling hit a roadblock as CMOS transistors started replacing MOSFETs - this has been

marked by a shift from micrometer (μm) scale to nanometer (nm) scale devices. There have been persistent efforts towards technological advancements, while utilizing Moore's Law as much as possible. For example, interesting breakthroughs have been made in the area by using self-aligned double gate MOSFET structures (FinFET) [5], which are projected to scale down to as small as 10nm. Improvements have been made to better the chip density as well, notably by three dimensional layering of active devices [6]. Researchers have also proposed a solution to enhance the energy efficiency of CMOS-based architectures [7] by intuitively utilizing voltage scaling techniques. Despite these leaps in technology and many similar ones, scaling down devices has kept on getting more and more difficult, each transition in scale bringing with it a new set of challenges and problems to be solved [8]. As the device dimensions are reduced, the traditional challenges posed by capacitance and resistance become more daunting, dielectric properties of materials interfere more, and physical properties like orientation of silicon lattice structures, wafer thickness and mechanical strain start having more effects on the device behavior. For instance, a cumulative effect of the above hurdles and the thermal Johnson–Nyquist noise [9] has been identified, which threatens to completely disrupt the correct behavior transistors at some point in the future. The greatest hindrance against the continuous downscaling of transistors is faced when the quantum mechanical properties (intrinsic to the semiconductor material) come into play. Such quantum effects are explored in [10], and to a greater extent in [11]. A compelling argument against the continuation of Moore's law can be drawn from the Heisenberg uncertainty, which establishes an upper bound on the constant miniaturization of semiconductor devices. Essentially, a transistor will stop behaving in a deterministic manner at a small enough scale, due to the quantum mechanical properties of matter. Given the above set of limitations (and many similar ones), it is reasonable to conjecture that Moore's Law is approaching its eventual demise. In a such a scenario it is only prudent to turn our attention to computational techniques which are not reliant on Moore's Law, thus ensuring undeterred and uninterrupted technological progress.

The Von Neumann architecture [12] was the first architecture that introduced the concept of stored-program computers - a development that sent ripples through the landscape of computational science over the following decades following its establishment. The architecture can be explained very simply by breaking it down into its modular components. The Von Neumann computer in Figure 1.1 can be interfaced with input and output devices, but its core computational unit is what interests us. It features a memory unit containing programs and data, which are fetched and processed by the central processing unit (CPU). The CPU itself consists of the arithmetic/logic unit (ALU) along with registers and a control unit consisting of an instruction register and program counter. The mathematical and engineering foundations for this architecture were subsequently laid in [13]. Von Neumann himself was acutely aware of some of the limitations of the architecture [14], and had pondered the possibility of using multiple computers in parallel in order to perform arduous computations. However, the major flaw in the Von Neumann architecture was the processor-memory bottleneck, as identified in [15]. As we know, there exists a sizable gap between processor and memory performance [16].

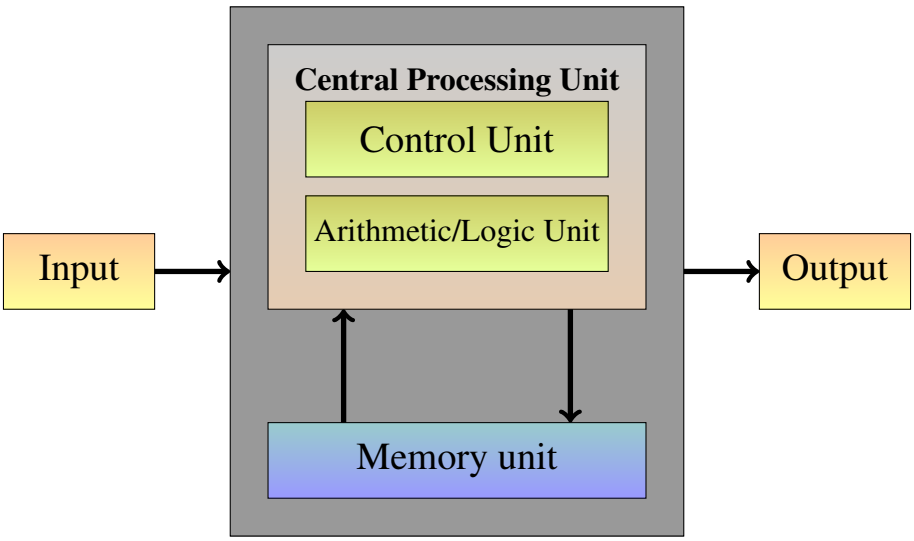


Figure 1.1: The Von Neumann architecture of a programmable computer

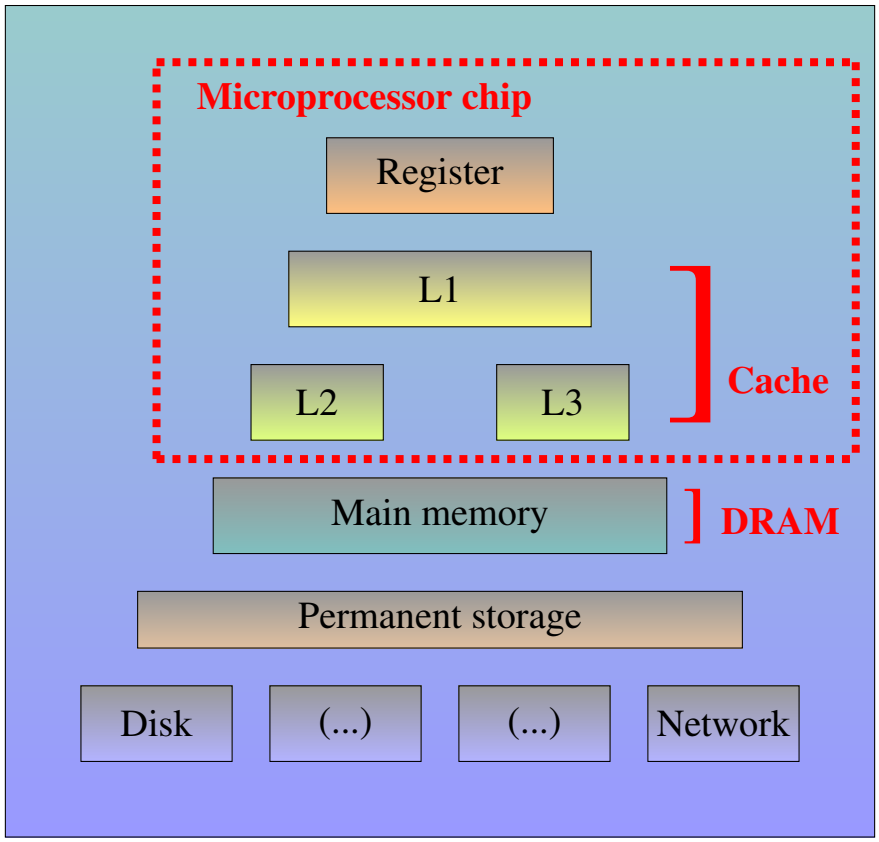


Figure 1.2: Memory hierarchy with on-chip cache memory

Additionally, the bus connecting the processor and the memory will always have a limited bandwidth, thereby throttling performance even further. Given this scenario, the Von Neumann architecture has ceased to be of much convenience in the current times. Despite the dynamic random access memory (DRAM) performance improving exponentially, it has been shown [17] that the processor-memory gap will also increase in an exponential manner, ultimately hitting a memory wall. One of the major approaches to tackle the bottleneck was the progress of on-chip cache memory as a component of the existing memory hierarchy [18], as shown in Figure 1.2. Cache memories boost performance mainly by pre-fetching instructions and data from the main memory (RAM) onto the microprocessor, thereby eliminating the need to use the limited-bandwidth bus in certain cases. However, issues like invalidation misses give rise to bad cache behavior [19] when trying to handle larger problems. With the development of faster microprocessors, there will

be greater demands placed on memory bandwidth, more specifically pin bandwidth [20], further choking the performance of future computers. While the Von Neumann architecture had once provided a simple and highly effective blueprint for computer design by abstracting away a great deal of engineering problems, it can no longer be utilized due to the plethora of disadvantages that have been exposed over time.

The eradication of the Von Neumann bottleneck has been targeted by a number of different approaches. A technique to negate the processor-memory bottleneck is to use the data-flow based programming paradigm. This model reduces the need to constantly fetch data by implementing data-driven instructions, i.e. a given instruction is only executed if and if its required operands are fetched at the end of a preceding instruction. Hence, data that would otherwise be fetched and overwritten (without being used) is no longer fetched, thereby reducing the effects of the bottleneck. The authors of [21] propose a processor design that employs the data-flow based model and is capable of performing highly parallel computations with low latency. One of the major pathways of research is provided by computation-in-memory (CIM) approaches. In [22], the authors propose a complete CIM-based parallel adder architecture which is shown be smaller, faster and more energy efficient than the state-of-the-art. A CIM architecture essentially gets rid of separate processor and memory units by using a crossbar. A crossbar is a circuit consisting of overlapping vertical and horizontal nanowires, with switchable memory elements placed at every junction where the wires intersect. Hence, memory can be stored at this location and computational operations are triggered and achieved by the communication network and external control circuitry. The communication network can be generalized as a two-dimensional mesh-based interconnection network as exhibited in [23], where the authors utilize this abstraction to implement a the highly capable Tile64 processor architecture powered by on-chip interconnects. It is worth noting that a crossbar architecture is maintained as the computational core of the system. Perhaps one of the most lucrative possibilities offered by CIM architectures is their capability to seamlessly accelerate parallel executions and algorithms. A crossbar-based template for such applications is introduced in [24] consisting of scheduling, placement and routing information - aptly named a “skeleton”. This solution maps the

data flow of some common parallel algorithms to crossbars and assesses the performance, space complexity and energy efficiency of the solution. A feasible alternative to having an end-to-end CIM architecture, a processor-in-memory (PIM) architecture can be used in synergy with a conventional Von Neumann computer to accelerate the execution of a given set of instructions [25]. An in-depth exploration of true potential of the PIM architecture is performed through the development of the Gilgamesh framework [26]. The framework accomplishes petaflops scale computing by implementing features like advanced task and data virtualization, adaptive resource management and object-based runtime management of system elements. At this point, we can observe that taking care of the Von Neumann bottleneck essentially entails the embedding of non-memory operations in the memory itself. In a similar vein of thought, logical operations are also performed in-memory [27] - a novel framework called logic-in-memory (LiM), which utilizes interpolation memory and a seed table with simple arithmetic logic is synthesized and shown be significantly better than conventional computing methodologies for a specific set of applications. One of the ways of achieving massively parallel computation had been through associative processors [28], which basically combines enormous arrays of traditional SIMD processors and memory (based on Von Neumann architecture). With the eventual demise of Moore's Law on the horizon, it would only be judicious to adapt such a simple, yet versatile, architecture to next generation technology like resistive memory devices [29]. A diligent survey of the above-mentioned literature reveals that crossbar-based architectures hold a great deal of promise towards yielding considerable breakthroughs in this field. At the heart of most crossbar circuits lie memristors or memristor-like devices [30], which we will discuss in greater details in subsequent sections.

Memristor - An overview of the fourth fundamental electrical element

In 1971, Leon Chua hypothesized that in addition to resistor, capacitor and inductance, there existed a fourth fundamental electrical element. This new element, which demonstrated a relation between flux linkage and electrical charge, was termed as a "memristor" [31] - an amalgamation

of memory and resistor. This name arose from the fact that a memristor's resistance varies with the net current flow through it and the resistance state would be maintained even if the device was disconnected from a power source, thereby giving it the property of non-volatile memory. It was further observed that memristors can be charge-controlled or flux-controlled. Given that we have *voltage* v, *current* i, *charge* q and *flux linkage* ϕ we can define the following:

$$d\phi(q) = \text{rate of change of charge with respect to the flux linkage} \tag{1.1}$$

$$dq(\phi) = \text{rate of change of flux linkage with respect to the charge} \tag{1.2}$$

Based on the above definitions, following expressions can be defined:

$$M(q) = \frac{d\phi(q)}{dq} \tag{1.3}$$

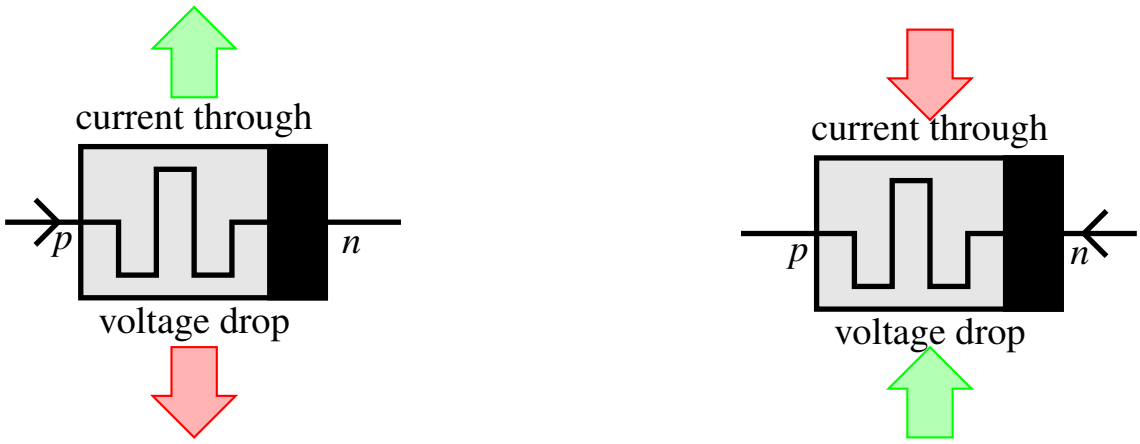
$$W(\phi) = \frac{dq(\phi)}{d\phi} \tag{1.4}$$

Where $M(q)$ is termed as the memristance (since it has dimensional equivalence with resistance) and $W(\phi)$ is termed as the memductance (since it has dimensional equivalence with conductance). Hence, the voltage across a charge-controlled memristor (at time t) and the current of a flux-controlled memristor (at time t) is characterized respectively as:

$$v(t) = M(q(t))i(t) \tag{1.5}$$

$$i(t) = W(\phi(t))v(t) \tag{1.6}$$

Hence we can see that the state of a given memristor can be changed by either changing its charge or its flux linkage, but not both. As we will see later, this has produced voltage-controlled and current-controlled memristive devices. Essentially, the internal resistance of the memristor is manipulated to change it's state. A memristor has two different terminals which correspond to positive (p) and negative (n) polarities.



(a) Transition from high resistance state to low resistance state

(b) Transition from low resistance state to high resistance state

Figure 1.3: State transition behavior of a memristor

The behavior of a memristor device can also be explained in a simpler manner. From Figure 1.3 we can see that the voltage drop across the memristor decreases and the current across it increases as it transitions from a high resistance state (HRS) to a low resistance state (LRS), when an external stimulus (current or voltage) is applied at the p terminal. In contrast, the voltage drop across the memristor increases and the current across it decreases when the as it transitions from a low resistance state (LRS) to a high resistance state (HRS), when an external stimulus (current or voltage) is applied at the n terminal. The HRS can be related to the highest possible resistance that the device can have and the LRS to the lowest possible state the device can have. Ideally, the HRS will be infinity (infinitely high resistance) and the LRS will be zero (no resistance at all). Therefore, we can say that the HRS is the OFF state and the LRS is the ON state. Hence, it can be said without doubt that the memristor device can be used as switch, and the transition from LRS to HRS (and vice versa) is thereby termed as "switching" behavior. A memristor in the LRS allows current flow across it, akin to a closed circuit or a turned-on switch. A memristor in the HRS allows no current across it, similar to an open circuit or a turned-off switch. This behavior is shown in Figure 1.4, and is limited by physical constraints, i.e. ideal behavior is not practically achievable.

(a) Low resistance state or *ON* state

(b) High resistance state or *OFF* state

Figure 1.4: Memristor in *ON* and *OFF* states

Memristors were further investigated and characterized using Lissajous figures by Chua [32]. A memristor's transitional behavior is identified as a pinched hysteresis loop, which underlies the special properties associated with a memristor. This makes it possible to explore the possibilities of using a memristor as an interesting non-linear element for computation, even though we will stick with its behavior as a switch for the scope of this document. Nearly 37 years after the conception of the memristor, a TiO_{2-x}-based discrete memristor device was fabricated [30] in 2008. However, it must be taken into account that this device did not use electromagnetism to change the internal state, even the "ideal" memristor requires a change in the electromagnetic field to change its internal state. It was recently argued [33] based on this fact that the missing memristor has not been found and is likely a physical impossibility – this is a debate that is currently active in the scientific community. The distinction between a device being an "ideal" memristor and it behaving like one is a small but important one. Nevertheless, such devices have tremendous potential to change the landscape of computer architecture and computer science in general. Let us now take a look at some interesting applications of memristive devices, which are commonly known was resistive RAM (RRAM or ReRAM) devices.

The authors of [34] have done a very thorough job of providing a balanced overview of the field of non-volatile memory devices, where both their applications solely as memory devices as well as their future as computational elements are examined. A number of different technologies are surveyed, and memristive devices are shown to be the only branch where full physical implementation has been achieved. Due their non-volatile property, ReRAMs are primarily being employed as memory elements in novel memory architectures. The authors of [35] have proposed a com-

pelling design for 3D stacked ReRAM caches – a design which has been shown to be at least as good as existing DRAM technology in most cases. However, use of memristors as a component for in-memory computation is arguably a more exciting area of research. A framework for performing stateful logic using memristor-based nanoscale crossbar circuits has been provided in [36]. While purely memristive circuits are shown to be rather intractable in case of large problems, it is possible to scale up the solution when each memristor is paired with a bi-directional diode. Even though a memristor can be used as a binary switch, its pinched hysteresis indicates that it also has analog properties that can be exploited. In [37] simple addition has been performed by combining this very property with cyclical programming. Here the output is not binary, but a conductance level at a fixed voltage. The crossbar fabric inherently supports parallel computation, since the state of all the memristive switches in a crossbar circuit can be changed in parallel quite trivially. This feature is utilized in [38], where the authors probe the application of memristive crossbar networks to the field of massively-parallel computing. A maze can be considered an abstraction for a wide variety of computational problems, and the authors have devised an algorithm to solve mazes, thereby extending the applicability of their approach a a number of different fields. At the same time, foundational research has been continuously been going on, as in [39]. It has been demonstrated that Boolean logic operations can be reliably implemented using a single memristor – the trick lies in applying the input bits periodically, in a sequential manner instead of applying them all at once. While this approach compromises the possibility of parallelism, the minimal size and high power efficiency more than make up for it. One of the impressive displays of using memristors at the core of parallel computation can be found in [40]. Not only has a design been proposed for performing parallel matrix multiplication, the scheme has also been shown to achieve a performance that is orders of magnitude better than existing methods. Finally in [41], a tentative yet promising solution has been suggested in order to unleash the potential of in-memory computation with memristors for big-data applications. This is a significant result as it shows that memristors are not a short-lived trend, but are capable of solving some of the largest and most complex computational problems that we are faced with today. Of course, there are many more

examples that validate the true potential of memristors to solve real-world problems – the above examples capture a reasonable picture of the more relevant sections of that spectrum.

CHAPTER 2: LITERATURE REVIEW

Circuit synthesis – A synopsis

As is known, synthesis is one of the major steps in circuit design. This can refer to the synthesis of network parameters or synthesis of the network topology. The synthesis of network parameters continues to be a critical aspect for analog circuit design. Early work in synthesis of network topology has been done in [42], where the authors had developed a symbolic circuit design tool SYN. The circuit constraints were provided to the tool, which has been shown to provide clear and concise results through incremental deduction. Perhaps one of the first overlaps between the field of algorithms and circuit synthesis can be found in [43]. In this case, the authors had utilized the branch-and-bound technique for constrained optimization of a two-stage CMOS-based operational amplifier design. With the passage of time and the advent of technology, the demand for more complex circuits were needed, thereby demanding more powerful automated synthesis tools. A knowledge-based approach towards such tools is presented in [43]. The OASYS tool equips the user with a hierarchical design methodology and a systematic exploration of the design space in order to produce circuits even for high performance systems. The fidelity of the circuits are as important as their complexity, which is why there is a necessity for tools that ensure reliable circuit synthesis even if a mismatch exists [44]. The set of constraints necessary for circuit synthesis can often be navigated by using innovative representations and search techniques [45] or by integrating existing techniques into new design methodologies [46]. Accurate synthesis for analog circuits can often be achieved by accounting for inherent stochastic effects (like parasitic capacitance) in a more energy-efficient manner [47] or by introducing error correction within the circuits themselves [48]. Synthesis techniques are equally useful in cases of re-configurable circuits [49], which enables rapid prototyping to aid in the advent of re-configurable device design. Circuit synthesis is not limited to analog or digital applications and has been used for synthesis of reversible [50, 51] and quantum [52, 50] circuits. This progress has facilitated the usage of quan-

tum computing principles to synthesize Boolean circuits [53]. Tools have also been developed to synthesize approximate circuits [54], as well as for the efficient synthesis of robust circuits [55]. Circuit synthesis at higher levels of abstraction [56] with a mathematically rigorous basis has also been explored. In contrast, stochastic search methods have also been effectively utilized [57] to synthesize novel architectures for a sub-class of circuits. Furthermore, automated design also been applied to obtain dynamic configurations for FPGA's [58]. The power of automated synthesis can also be extended to come up with circuits that not only have lower space requirements, but also capable to delivering optimal or near-optimal power consumption [59].

One of the earlier developments if the field of logic synthesis was demonstrated through the now well-known Karnaugh maps [60]. A symbolic method for synthesizing combinational circuits [61] was demonstrated which yielded functional, but not necessarily optimal circuits. This was soon followed by the work by McCluskey [62], which laid the groundwork for subsequent synthesis of combinational circuits executing Boolean logic. Multiplexers were identified [63] as modular components that could be used for automatically synthesizing combinational circuits – a development that would later be utilized widely in the field of circuit design as well as automated circuit synthesis. The advent of both computer hardware and software yielded one of the earliest circuit synthesis tools [64]. While MACDAS was capable of producing functional combinational circuits, there was still a necessity for tools what could construct sequential circuits. This gap was filled by SIS [65, 66], which truly became one of the landmark tools in automated circuit synthesis. Its utility was aggregated by the fact that SIS incorporated a number of optimization techniques within the synthesis process – this resulted in combinational circuit designs which were not only functional but also optimal or near-optimal. Once such potent tools were developed, the next significant undertaking largely involved the improvement of the said tool. Decompositional logic synthesis [67] and multi-level logic synthesis [68] were some of the numerous approaches that were probed in an attempt to further the contemporary automated synthesis techniques. The power of Boolean satisfiability solving (SAT solving) was applied [69] for producing asynchronous circuits, which was soon followed by the development of the Tangram framework [70]. The framework was then

extended as the BALSA system [71] – a tool which would later be used during the development of self-timed ARM processor cores. The problem of synthesizing sequential synchronous logic circuits has been attacked in a similar fashion [72], with the added objectives to optimize cycle time. The research in this direction yielded circuits with lower delay and synthesis algorithms which were compatible with the existing technologies at the time. The growing interest for research in the field saw produced some interesting applications like the synthesis of circuits with concurrent error detection with a reduced area requirement [73], as well as development of larger, more complex pass transistor-based circuits using Boolean decision diagrams (BDDs) [74]. Similar ideas had been explored at an earlier time, with goals to minimize power consumption [75]. Among many other publications, the significance of representation and manipulation of combinational circuits as abstract data structures was consolidated in [76].

Existing memristive and resistive switching devices

The physical switching devices form the basis for realization of the synthesized logic and architecture. A brief summary of notable memristive and resistive switching devices has been provided in Table 2.1. There also exist a number of comprehensive reviews of such devices in current literature. The authors of [77] explore the resistive switching phenomena in transition metal oxides. While [78] takes an in-depth look at a sub-class of the devices developed through oxidation-reduction (redox) methods and their inherent nanoionic mechanisms, [79] utilizes the NEXAFS tool to gain a more profound insights into the behavior of TiO_2-based devices specifically. The fabrication challenges often arise from an intent to create smaller devices – reducing the space requirement for the the switching mechanisms and actual materials both play a crucial role.

A broad perspective of existing technologies and devices is given by [109, 110]. The nature of the sub-class of thin film resistive switching devices has been thoroughly studied in [111], and the role of contemporary devices for primarily neuromorphic computing has been considered in [112].

Table 2.1: Overview of switching devices

Year	Device	Materials	Structure	Width
2006	[80]	TiO_2/TiN	Thin film	50nm
2007	[81]	Cu/SiO_2	Programmable metallization cell	100nm
	[82]	Ti/ZrO_2/Pt	Metal–insulator–metal	250μm
	[83]	Cr/$SrTiO_3$	Metal–insulator–metal	500nm
2008	[84]	ZrO_2/Zr+	Sandwich	50-800μm
	[85]	Si/a-Si$\times$Ag	Core/shell nanowires	20nm
	[86]	Cu/ZrO_2:Cu/Pt	Sandwich	3-20μm
	[30]	TiO_2/TiO_{2-x}	Thin film	3nm
2009	[87]	NiO/*n*–doped Si	Metal—insulator—semiconductor	160nm
	[88]	Si/SiO_2/Ti/Pt/TiO_2/Pt	Thin film	50nm-5μm
	[89]	Ti/Pt/TiO_2/Pt	Electroformed substrate	70nm
	[90]	Au/ZnO/stainless steel	Thin film	35nm
2010	[91]	Pt/TaO_x/Ta	Stack	100μm^2 (area)
	[92]	Ag/a–Si/p–Si	Filament	50nm
	[93]	Al/GO/ITO	Metal—insulator-–metal	30nm
	[94]	Fe-doped $SrTiO_3$	Thin film	35nm
	[95]	Pt/TiO_2/Pt	Metal—insulator-–metal	140nm
2011	[96]	ZnO	Thin film	82μm
	[97]	TiN/Al_2O_3/Pt	Metal—insulator-–metal	50nm
	[98]	p–Si/HfO_2/p–Si	Filament	50nm (HfO_2)
2012	[99]	Ag/a-Si/Pt	Filament	100nm
	[100]	Pd/Ta_2O_{5-x}/TaO_y/Pd	Thin film	0.5-2μm
	[101]	SiO_x	Thin film	15-120nm
2013	[102]	SiO_2/Si	Cross-point	8nm
2015	[103]	Ti/ZnO/Mo	Metal—insulator-–metal	50-500nm
2017	[104]	Pt/Ta_2O_{5+x}/Ta	Metal—insulator-–metal	20nm
2018	[105]	Pt/HfO_2/TiN	Filament	300nm
	[106]	Cu/MoS_2/W_2N	Thin film	200 nm
	[107]	HSQ	Metal—insulator-–metal	200μm-800μm
2019	[108]	Ag/HfAlOx/Pt	Metal—insulator-–metal	136nm

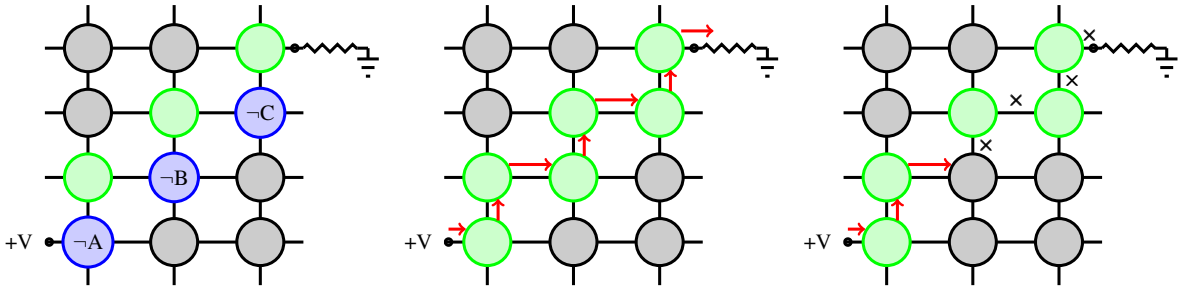
Memristors are are capable of showing controllable, yet probabilistic, switching behavior in [113] – an interesting phenomenon of critical importance for practical stochastic computing frameworks.

Redox devices are revisited in [114], where their applicability with respect to Boolean function realization is inspected. Finally, a rigorous survey of current resistive switches manufactured using metal oxides is presented in [115]. All in all, this section provides a broad, surface-level picture of some compelling memristive and resistive switching devices.

Crossbar synthesis for computation using sneak paths

In any given graph, a path is said to exist between two nodes if there exists a set of edges that connects the two nodes. In other words, existence of a path implies that one node is reachable from the other. In a crossbar, a connection might or might not exist between two nanowires. If a connection *does* exist, then a current flow introduced in one of the nanowires at time t will be observable at the other nanowire at a time $(t + \varepsilon)$, where ε is the short amount of time it takes for the current to flow from one point to another. When an undesired path exists, the problem is termed as a “sneak" path. However, the authors of [116, 117] have managed to turn this potential drawback into a computational mechanism. A Boolean formula is mapped into a crossbar with the following properties:

- There must exist memristors in the crossbars corresponding to literals in the Boolean formula.
- The memristors must be switched ON or OFF, depending on the input instances.
- A current flow must be introduced in one of the nanowires, and the index of the nanowire is constant for a given formula.
- If the Boolean formula evaluates to TRUE for a given input instance, then a current flow must be observable at a predefined nanowire. The index of this output nanowire must be constant for a given Boolean formula.



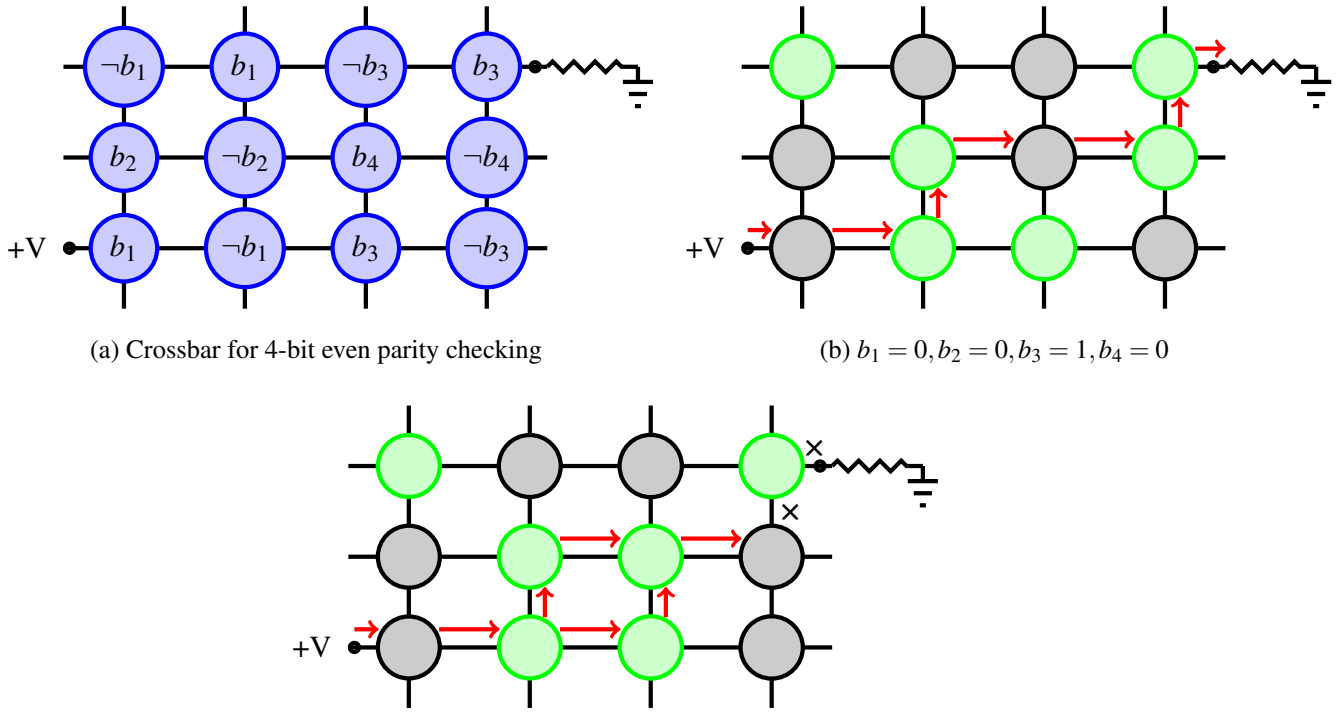
(a) Crossbar design (b) A=0, B=0, C=0 (c) A=0, B=1, C=0

Figure 2.1: Crossbar implementing $\neg A \wedge \neg B \wedge \neg C$ with two input instances

- If the Boolean formula evaluates to FALSE for a given input instance, then a current flow must not be observable at a predefined nanowire. The index of this output nanowire must be constant for a given Boolean formula.

The authors of [116] have mapped Boolean formulas in the negation normal form (NNF). An example of a crossbar implementing the Boolean formula $\neg A \wedge \neg B \wedge \neg C$ has been provided in Figure 2.1. The blue circles represent memristors which assume the value of literals, the green circles represent memristors in the ON state and the grey circles represent memristors in the OFF state. In Figure 2.1(b) and 2.1(c), the red arrows represent current flow and the crosses represent lack of current flow. However, this method produces crossbars that are very large and therefore inefficient with respect to space constraints. For this reason, the authors of [118] have leveraged the power of formal methods to synthesize compact memristors. The crossbar is synthesized as a symbolic mapping matrix, which must adhere to certain logical constraints. These constraints specify the flow-transition behavior of the crossbar, as well as its size constraints. The synthesis is achieved in a smaller number of computational steps than existing methods and produces more compact crossbars. One of the designed crossbars implements 4-bit even parity check, as shown in Figure 2.2, where the blue circles represent memristors which assume the value of literals, the green circles represent memristors in the *ON* state and the grey circles represent memristors in the OFF state. In Figure 2.2(b) and 2.2(c), the red arrows represent current flow and the crosses

represent lack of current flow.



(a) Crossbar for 4-bit even parity checking

(b) $b_1 = 0, b_2 = 0, b_3 = 1, b_4 = 0$

(c) $b_1 = 0, b_2 = 0, b_3 = 1, b_4 = 1$

Figure 2.2: Crossbar implementing 4-bit parity checking with two input instances

The automated synthesis of compact crossbars opened up the possibility of implementing them in hardware. In [119], a crossbar implementing a full adder has been realized. The resistive switches were implemented using a IBM 65nm 10LPe process – each device was fabricated using SiO_2, TaN/Ta, electroplated Cu and HfOx. The design of the fabricated circuit is provided in Figure 2.3. The blue circles represent memristors which assume the value of literals, the green circles represent memristors in the ON state and the grey circles represent memristors in the OFF state. In Figure 2.3(b) and Figure 2.3(c), the red arrows represent current flow and the crosses represent lack of current flow. The mechanism using sneak paths has been used in [120] to perform fast Boolean matrix multiplication using crossbar circuits. A memristor is usually bi-directional, i.e., an ON memristor allows current flow through itself regardless of the terminal the flow is introduced in.

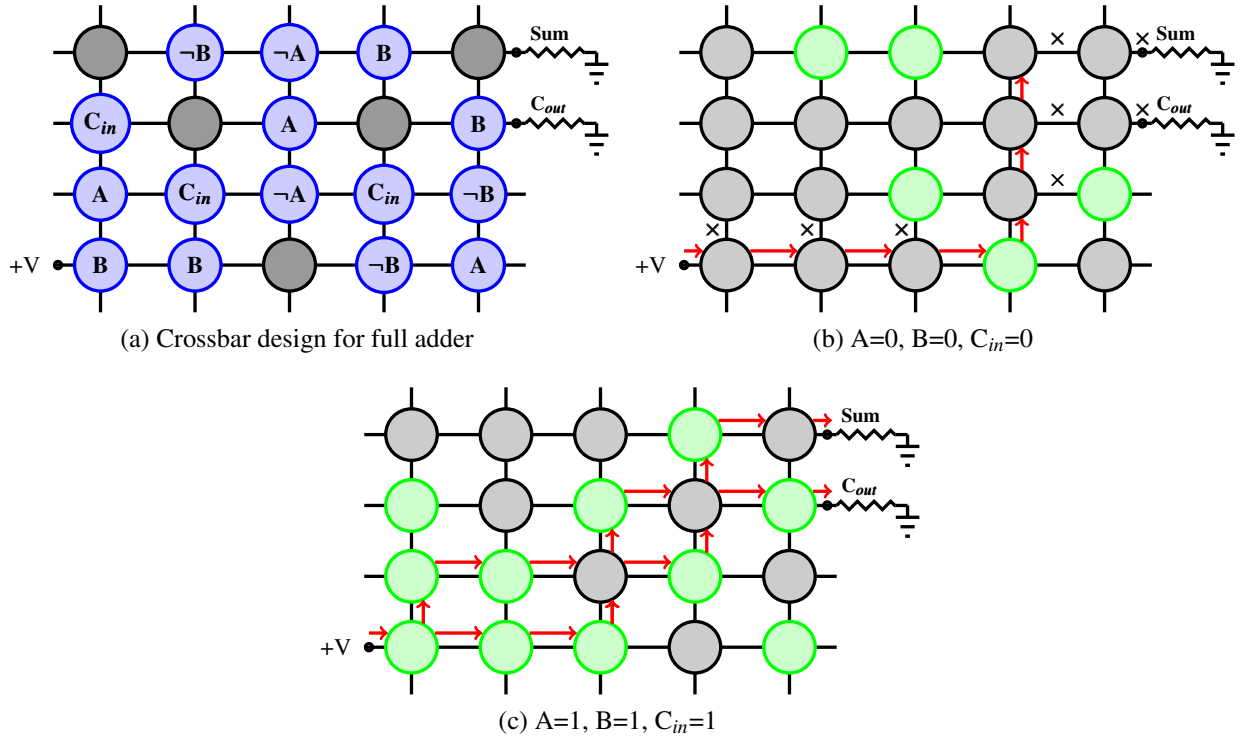


Figure 2.3: Crossbar implementing a full adder with two input instances

However, this is a property that can sometimes generate unwanted sneak paths. In such a scenario, a diode can be connected in series with the memristor to ensure that current flows in a single direction. This combined device is termed as a rectifying memristor, and they can be used to synthesize more compact crossbars that are capable of performing intensive computations [121] like Boolean matrix multiplications. This is a topic that we will explore in greater depth in some of the following sections of this document.

CHAPTER 3: MOTIVATION

From the background work, we can observe that there exists a definite need to eliminate or circumvent the Von Neumann bottleneck for post-Moore's Law computing paradigms. Based on the literature review section, we are aware that there are a significant number of resistive switches that can serve as nanoscale switches to be deployed in emerging architectures for both in-memory computation and memory systems. The conjunction of these two developmental factors has yielded a number of novel research topics, ripe for excavation and exploration. The crossbar circuit topology has made its way into mainstream technological progress [122, 123]. Moreover, significant development has been made towards efficient fabrication of the crossbar circuits [124, 125, 126]. A combination of the above developments has led to the production of high-performance memory devices [127, 128] on the crossbar fabric. While all these advancements have resulted in performance and scalability improvements, the Von Neumann bottleneck has remained a persistent detractor of further gains in computing power.

One of the solutions that would effectively deal with the bottleneck is the notion of in-memory computing – the data is stored on the same physical device that performs computation on it. The principles of flow-based computing provide a novel and transformational framework to perform in-memory computing with memristor-based crossbar networks. There exist multiple problems, as outlined in the abstract, which have potential to make considerable impact upon the current state-of-the-art in this area of research. Hence, making non-trivial contributions towards the development of in-memory computation on memristor-based crossbar circuits serves as compelling motivation for this body of work.

CHAPTER 4: AUTOMATED SYNTHESIS OF LARGE SCALE CROSSBARS WITH REDUCED ORDERED BINARY DECISION DIAGRAMS

Crossbars as abstractions

In order to synthesize crossbar designs successfully, it is inefficient and cumbersome to stick to electrical models of crossbar circuits. Hence, we to encapsulate crossbar circuits as abstractions (that can be easily manipulated to aid in synthesis procedures) as follows:

Definition 1. CROSSBAR *A (memristor-based) crossbar is a 3-tuple* $\mathbb{C} = (\mathbb{M}, \mathbb{W}_r, \mathbb{W}_c)$ *where*

- $\mathbb{M} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l1} & m_{l2} & \cdots & m_{ln} \end{pmatrix}$ *is a two-dimensional array of memristors with l rows and n columns, where* $m_{ij} \in [0,1]$ *denotes the state of the memristor (ON or OFF) connecting row i with column j.*
- $\mathbb{W}_r = \{r_1, \cdots, r_l\}$ *is the set of horizontal nanowires such that the wire* r_i *provides the same input voltage to every memristor in row i.*
- $\mathbb{W}_c = \{c_1, \cdots, c_l\}$ *is the set of vertical nanowires such that the wire* c_j *provides the same input voltage to every memristor in row i.*

The memristor $m_{ij} = 0$ is said to be in the high-resistance state or OFF state and $m_{ij} = 1$ denotes a memristor in the low-resistance state or ON state.

Definition 2. CROSSBAR DESIGN *A crossbar design* $\mathscr{D}(\mathbb{M})$ *maps each memristor* m_{ij} *in the crossbar* $\mathbb{M}$ *to one of the following: an input Boolean variable* $v_1, \cdots, v_n$*, its negations* $\neg v_1, \cdots, \neg v_n$ *or the logical constants True or False.*

For a crossbar with l rows, n columns and v different input variables, each memristor can be

mapped to $2v+2$ different values. Hence the number of possible crossbar designs is as large as $(n \times l)^{2v+2}$. As mentioned previously, “sneak" paths have been used as design primitives for suitable designed memristive crossbars. For a given crossbar design corresponding to a Boolean formula and a given set of values for the input variables, a current flow can be introduced in one of the nanowires. The index of this nanowire is determined by the design specifications, and is often the top or bottom row (horizontal nanowire) or the first column (vertical nanowire), of the crossbar. A non-negligible voltage is observed at the output nanowire if and only if the Boolean formula evaluates to true for the given set of inputs. Conversely, a negligible voltage is observed at the output nanowire if and only if the Boolean formula evaluates to false for the given set of inputs. The negligible and non-negligible voltage levels represent the logical constants true and false, respectively. The index of the output nanowire is once again determined by the design specifications, and it is often the top or bottom row (horizontal nanowire) of the given crossbar.

Binary Decision Diagrams (BDDs) and Reduced Ordered Binary Decision Diagrams (ROBDDs)

A Boolean Decision Diagram (BDD) [129, 130] is a rooted directed acyclic graph that consists of two types – terminal nodes representing the logical values 0 (for false) and 1 (for true) and non-terminal Boolean nodes labeled by Boolean variables. Each decision node has a “high” edge and a “low” edge. The high edge represents the case when the Boolean variable associated with the node has a value of 1. Similarly, the “low” edge represents the case when the Boolean variable associated with the decision variable has a value of 0. For a given value of a variable, only one of the edges from the decision variable can exist at a time. Hence, for a given set of inputs, there exists a path from the root node to one of the terminal nodes. The terminal node represents the output of the given Boolean function with respect to the given set of inputs. If a path exists from the root node to the zero terminal, then the Boolean formula evaluates to false. Similarly, if a path exists from the root node to the one terminal, then the Boolean formula evaluates to true. An Ordered Binary Decision Diagram (OBDD) is a Binary Decision Diagram where the

variables appear in the same order from the root node to one of the terminal nodes, regardless of the input set that is provided to it. Variable ordering affects the size of a BDD. The "correct" or optimal variable ordering can result in a BDD of optimal size, an "incorrect" or non-optimal variable ordering can cause the size of a BDD to explode. While an optimal variable ordering is not always necessary, a good variable ordering which yields in a BDD of reasonable size is rather desirable. A Reduced Ordered Binary Decision Diagram (ROBDD) is an ordered binary decision diagram where isomorphic sub-graphs have been merged and nodes whose children are isomorphic to mean other have also been removed. The transformation of an OBDD to a ROBDD returns a decision diagram with a lower number of nodes at the different levels, or decision diagrams which have a fewer number of levels overall. The effectiveness variable ordering and reduction varies with the structure and complexity of the Boolean formula being represented by the ROBDD. Due to the fact that ROBDDs are canonical for a given function and a specific variable ordering, they have found significant applications in functional equivalence checking and functional technology mapping. A number of different libraries like CUDD [131], BuDDy, etc. exist, which facilitate ROBDD-based research and application development with greater ease.

Inductive mapping of ROBDDs to crossbar circuits

A ROBDD for a Boolean formula ϕ is a directed acyclic graph G_ϕ where the nodes are naturally divided into levels such that all outgoing edges from nodes at level i only connect to nodes at levels $i+1$ or higher. The subgraph $G_\phi[i:j]$ of a ROBDD containing all nodes from level i to level j is mapped into a crossbar $\mathbb{M}(G_{phi}[i:j])$ using induction.

Base Case: The smallest entity in our design approach is a subgraph $G_\phi[i:i]$ with nodes at the same i^{th} level in the ROBDD.

As shown in Figure 4.1, let a single level $G_\phi[i:i]$ of a Boolean Decision Diagram contain c nodes $x_0, x_1, \ldots, x_c$. Each node x_i as at most two children: if the variable x corresponding to the node x_i

is true, the child is x_i^r. On the other hand, if the variable x is false, the child of the node x_i is x_i^l. The logical value of the children coupled with the logical value of the variable labeling this node produces the logical value computed by the nodes at this level.

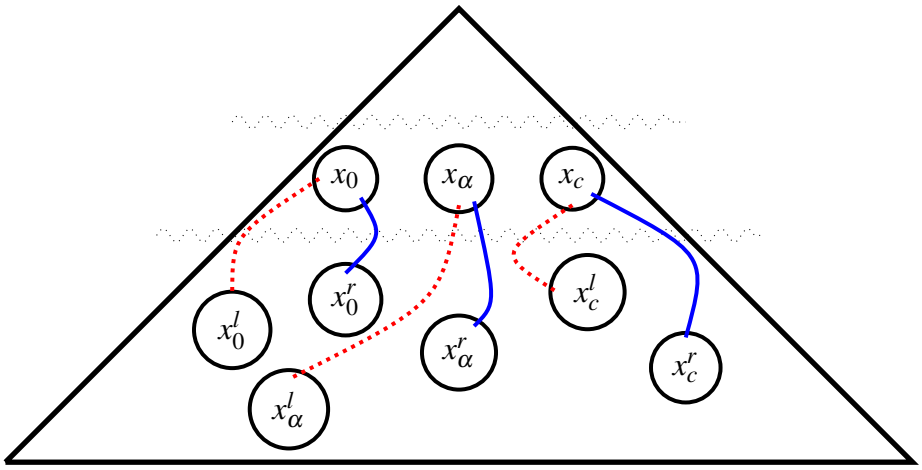


Figure 4.1: Nodes at a single level of a ROBDD and their children

Since the nodes $x_0^l, x_0^r, \ldots, x_c^l, x_c^r$ are at levels lower than the nodes $x_0, x_1, \ldots, x_c$, these nodes compute fucntions that are independent of the variable labeling the i^{th} level or of variables in nodes at higher levels. The single level of a Binary Decision Diagram with c nodes is mapped to a crossbar with approximately $3c$ rows and c columns. From there on, the induction will proceed on the number if levels H in the fragment $G_\phi[i:j]$ of the Boolean Decision Diagram. The variables are assigned *TRUE* (*ON* memristor) or *FALSE* (*OFF* memristor) values on the basis of a given input instance.

Inductive Hypothesis: For every subgraph $G_\phi[i:j]$ of a ROBDD containing all nodes from level i to level j such that $j \geq i$ and $j - i \leq H$, there exists a crossbar $\mathbb{M}(G_\phi[i:j])$ that contains horizontal nanowires $h_{v_0^{i-1}}, h_{v_1^{i-1}}, \ldots, h_{v_{c-1}^{i-1}}$ computing the value of the Boolean formula corresponding to the nodes $v_0^i, v_1^i, \ldots, v_{c-1}^i$ given that the values corresponding to the nodes $v_0^j, v_1^j, \ldots, v_{c'-1}^j$ are available on the corresponding input nanowires. Figure 4.2 demonstrates a crossbar that computes $G_\phi[i:j]$

along with its input and output nanowires. The inductive hypothesis assumes that the subgraph $G_\phi[i:j]$ can be mapped to a crossbar with the inputs (corresponding to level j of the ROBDD) being fed along the lower rows and the outputs (corresponding to level i of the ROBDD) leaving the crossbar along the higher rows.

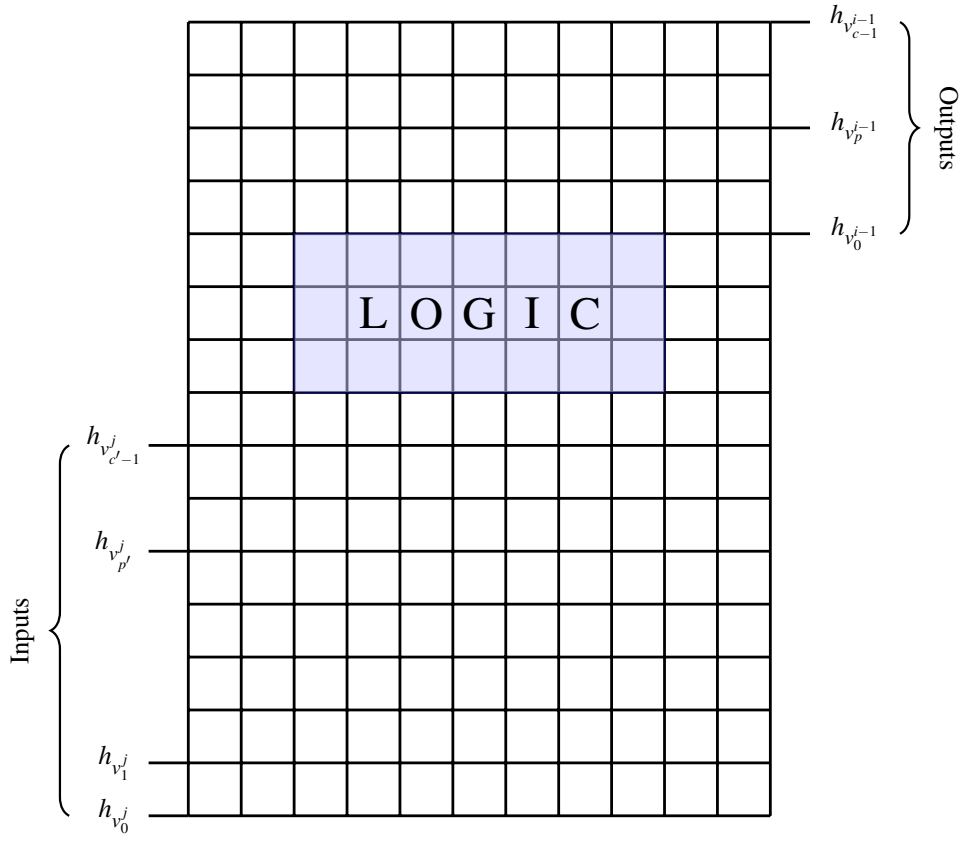


Figure 4.2: Mapping a subgraph to a crossbar, as a basis for the inductive hypothesis

Induction: Consider a subgraph $G_\phi[i:j]$ containing all nodes from level i to level j ($j > i$ and $j - i = H$) of a ROBDD G_ϕ for the Boolean formula ϕ. First, a level k between i and j is chosen. The subgraph $G_\phi[i:j]$ is divided into subgraphs $G_\phi[i:k]$ and $G_\phi[k+1:j]$ with directed edges $(u_0^k, v_0^k), (u_1^k, v_1^k), \ldots, (u_{c-1}^k, v_{c-1}^k)$ from the subgraph $G_\phi[i:j]$ to $G_\phi[k+1:j]$. Similarly, $(u_0^{i-1}, v_0^{i-1}), (u_1^{i-1}, v_1^{i-1}), \ldots, (u_{d-1}^{i-1}, v_{d-1}^{i-1})$ are directed edges coming into $G_\phi[i:k]$ and (u_0^j, v_0^j), $(u_1^j, v_1^j), \ldots, (u_{e-1}^j, v_{e-1}^j)$ are directed edges coming out of $G_\phi[k+1:j]$.

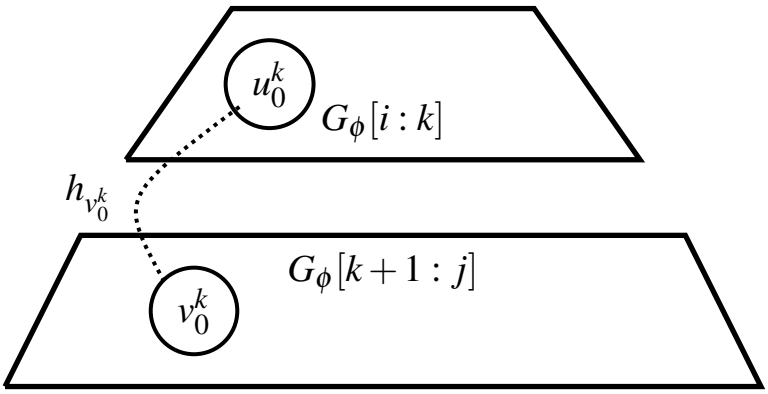
By the inductive hypothesis, for the subgraph $G_\phi[k+1:j]$, there exists a crossbar $\mathbb{M}(G_\phi[k+1:j])$ that contains horizontal nanowires $h_{v_0^k}, h_{v_1^k}, \ldots, h_{v_{c-1}^k}$ computing the value of the Boolean formula corresponding to the nodes $v_0^k, v_1^k, \ldots, v_{c-1}^k$ given that the values corresponding to the nodes $v_0^j, v_1^j, \ldots, v_{e-1}^j$ are available on the corresponding input horizontal nanowires. Similarly, the the inductive hypothesis, there exists a crossbar $\mathbb{M}(G_\phi[i:k])$ that contains horizontal nanowires $h_{v_0^{i-1}}, h_{v_1^{i-1}}, \ldots, h_{v_{d-1}^{i-1}}$ computing the value of the Boolean formula corresponding to the nodes v_0^{i-1}, $v_1^{i-1}, \ldots, v_{d-1}^{i-1}$ given that the values corresponding to the nodes $v_0^k, v_1^k, \ldots, v_{c-1}^k$ are available on the corresponding input horizontal nanowires. As shown in Figure 4.3, the crossbar corresponding to the subgrpah $G_\phi[i:j]$ can be obtained by merging smaller crossbars corresponding to the subgraphs $G_\phi[k+1:j]$ and $G_\phi[i:k]$ of the Boolean Decision Diagram. The lower crossbar may produce outputs in a different order than the inputs expected by the upper crossbar but permutations of rows (or columns) do not change the computation being performed by a crossbar; hence the rows of one crossbar can be rearranged to meet the desired rows of the other crossbar.

Theorem 1. The number of rows R(N) and the number of columns C(N) of a memristive crossbar $\mathbb{M}(G_\phi)$ that computes the Boolean formula ϕ corresponding to a ROBDD G_ϕ is linear in the number of nodes N of the ROBDD G_ϕ.

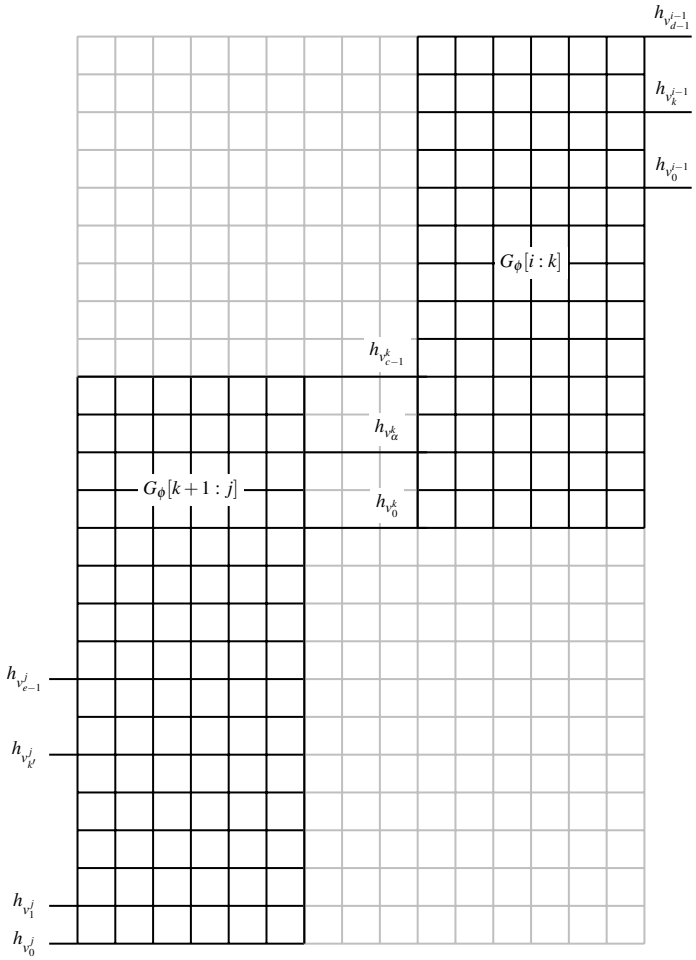
Proof. We will show that $R(i) \leq 3i$ for all i using mathematical induction where i is the number of nodes between two levels of a ROBDD.

Base Case: An upper bound on the number of rows $R(i)$ for BDDs with one node can be computed as:

- $R(|G_{True}|) \leq 3$.
- $R(|G_{False}|) \leq 3$.
- $R(|G_x|) \leq 3$.
- $R(|G_{\neg x}|) \leq 3$.



(a) The subgraphs $G_\phi[i:k]$ and $G_\phi[k+1:j]$ of the BDD with one of the edges (u_0^k, v_0^k).



(b) Inductive synthesis of the crossbar for $G_\phi[i:j]$ from the crossbars corresponding to subgraphs $G_\phi[k+1:j]$ and $G_\phi[i:k]$. The nanowire $h_{v_0^k}$ carries the value corresponding to node v_0^k in the crossbar for $G_\phi[k+1:j]$ to the crossbar for $G_\phi[i:k]$.

Figure 4.3: Inductive synthesis of memristor crossbar circuits based on ROBDDs

Hence, $R(1) \leq 3$.

Inductive Hypothesis: Assume that $R(i) \leq 3i$ for all $i < N$, where i is the number of nodes between two levels of a ROBDD, say $G_\phi[i:j]$.
Inductive Step: Consider the number of rows R(N) corresponding to the number of nodes between two levels of a ROBDD, say $G_\phi[i:j]$.

$$\begin{aligned}
R(N) &= R(|G_\phi[i:j]|) \\
&\leq R(|G_\phi[i:k]|) + R(|G_\phi[k+1:j]|) \text{ (by design of the crossbar)} \\
&\leq 3|G_\phi[i:k]| + 3|G_\phi[k+1:j]| \text{ (by inductive hypothesis)} \\
&= 3(|G_\phi[i:k]| + |G_\phi[k+1:j]|) \\
&= 3N \text{ (since, } |G_\phi[i:k]| + |G_\phi[k+1:j]| = |G_\phi[i:j]| = N)
\end{aligned}$$

By symmetry, the number of columns $C(N)$ is also linear in the number of nodes of the ROBDD. The identical argument is omitted for the sake of brevity. □

Next, we show the step-by-step mapping of a given ROBDD to a memristive crossbar circuit. In this case, the ROBDD represents the most significant bit of the C_{out} of Boolean addition between two 3-bit binary numbers, A and B. The least significant bit of the variable A is referred by A_0, the most significant bit is referred by A_2, and the middle bit is A_1. The same naming convention is followed for the variable B, as well. The ROBDD with demarcated levels is presented in Figure 4.4. In case of the crossbar, a current flow is introduced at the first column and an output flow (or lack thereof) is observed at the bottom row. The blank crossbar (without any memristors placed at the intersections of vertical and horizontal nanowires) is presented in Figure 4.5, along with the level-by-level mapping of the ROBDD to the crossbar. As can be observed, the nodes are mapped to the crossbar first, then permanently *ON* memristors are added to represent the edges between the nodes. The switches corresponding to the nodes in level 0 of the ROBDD are mapped in Figure 4.5(b).

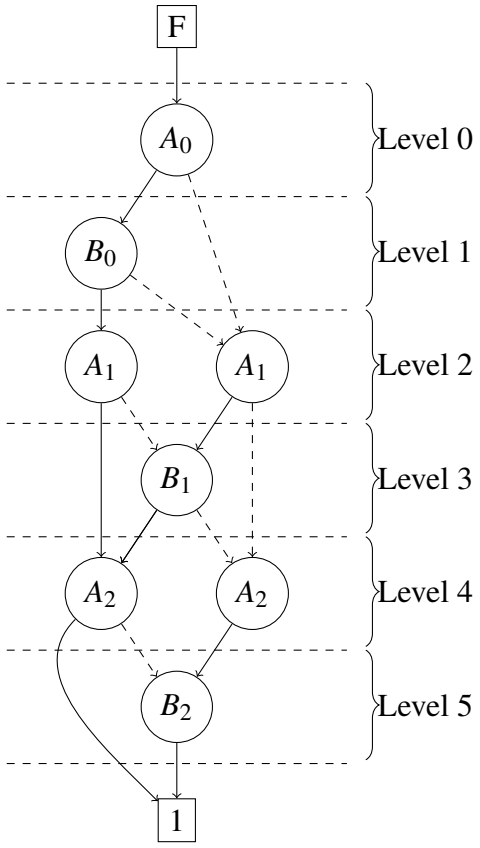
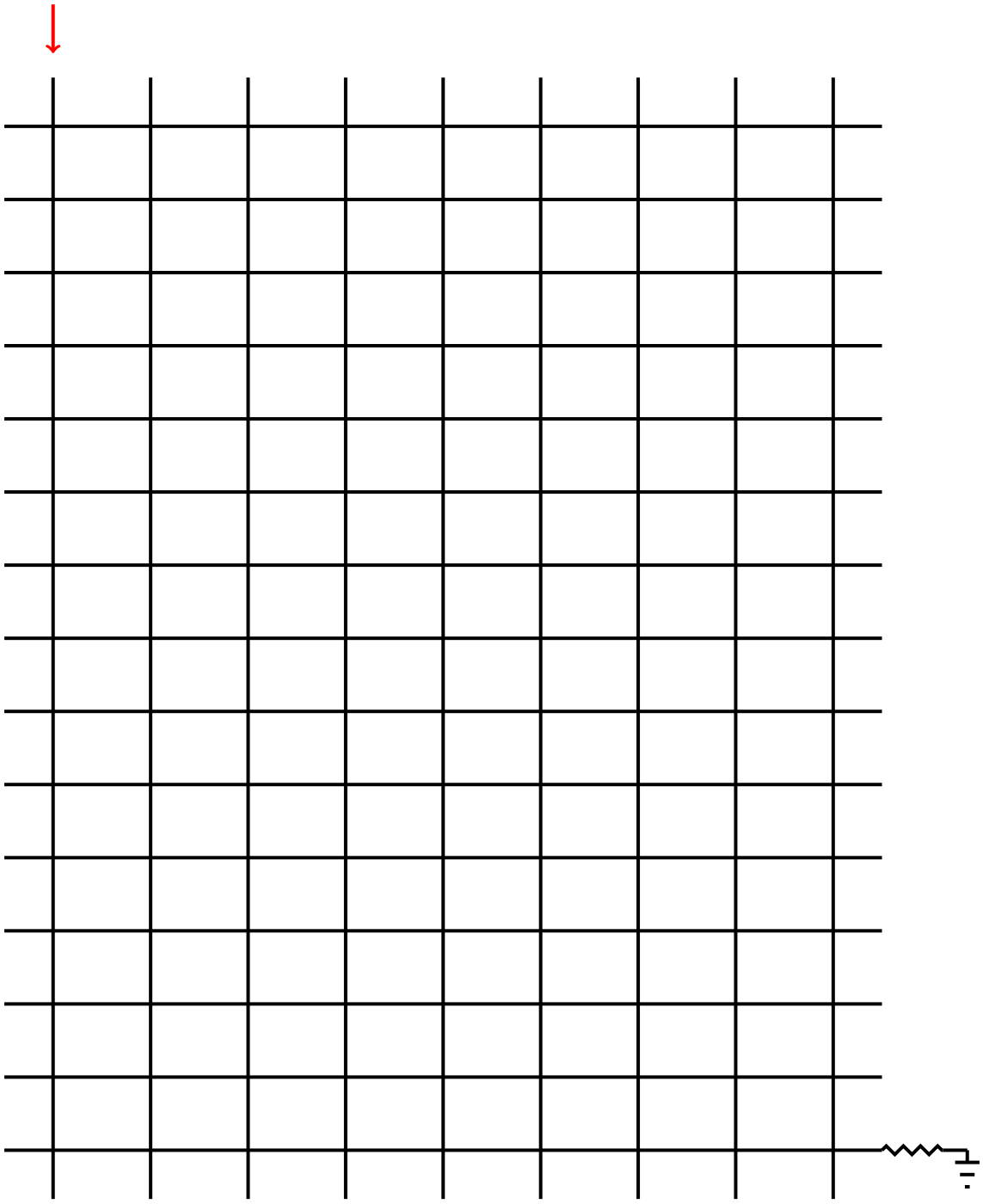


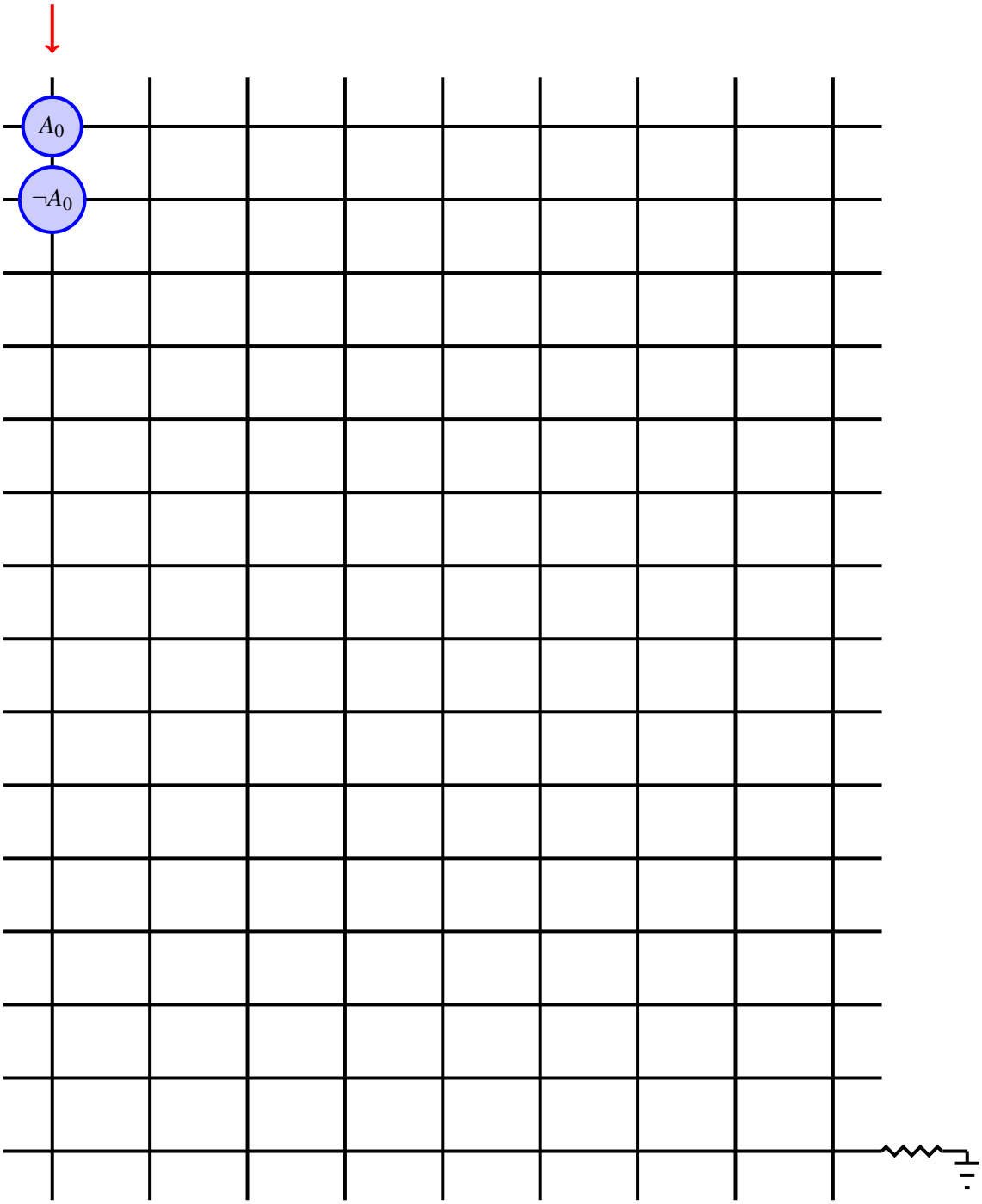
Figure 4.4: ROBDD representing the most significant bit of 3-bit binary addition

In a similar fashion, the switches corresponding to the nodes in level 1 are added to the crossbar in Figure 4.5(c), switches corresponding to the nodes in level 2 of the ROBDD are added in Figure 4.5(c) and so on, till Figure 4.5(g), where we add the switches corresponding to the nodes in level 5 of the ROBDD. Finally, the permanently *ON* memristors corresponding to the directed edges of the ROBDD are added in Figure 4.5(h). Once these memristors have been placed, the rest of the junctions are filled with permanently OFF memristors, as shown in Figure 4.5(i). In this way, there exists a path from the first column to the bottom row if and only if the Boolean formula evaluates to true and the memristors are turned to the *ON* or OFF states corresponding to their binary values. We have presented this approach in [132].

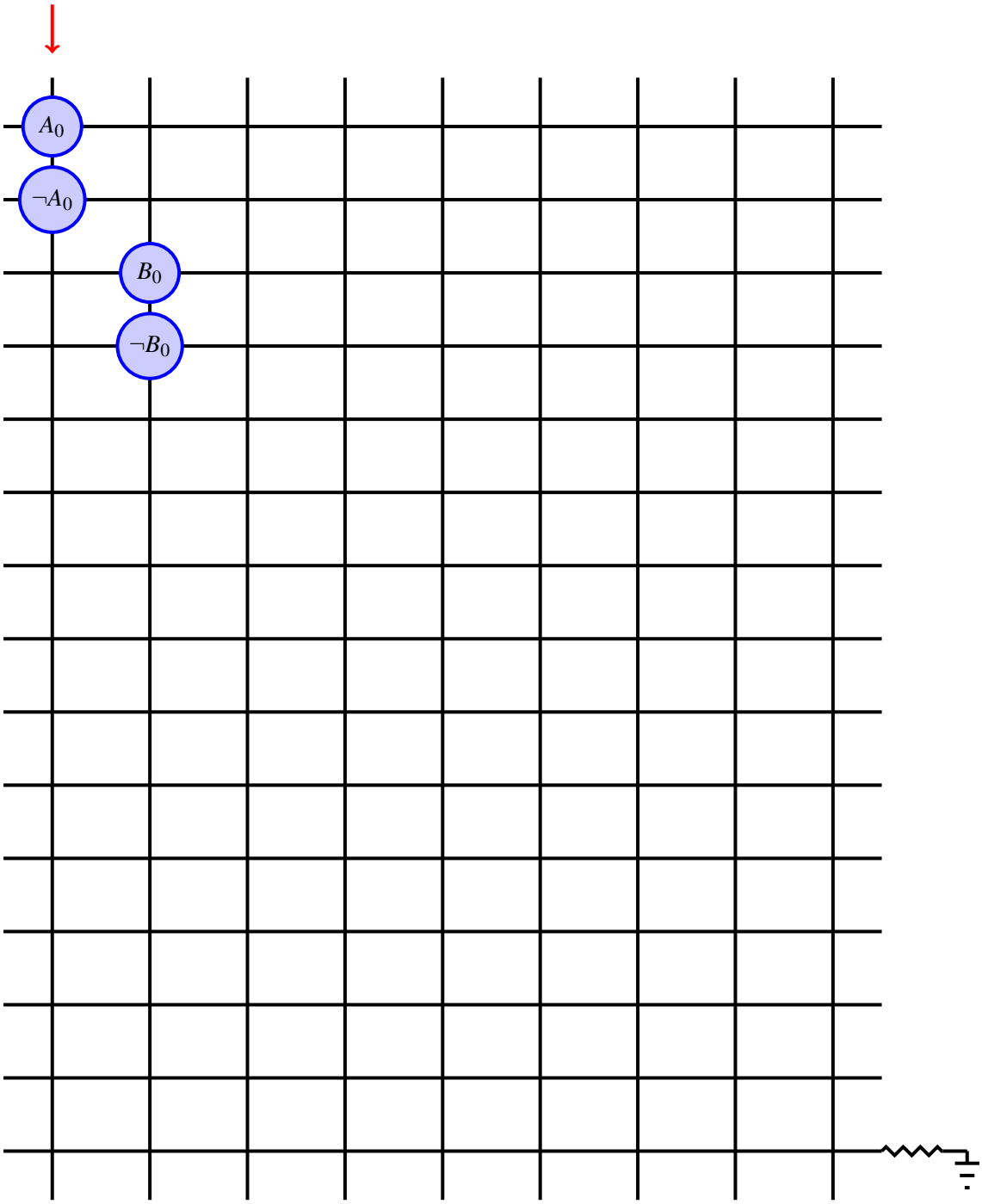


(a) Blank crossbar

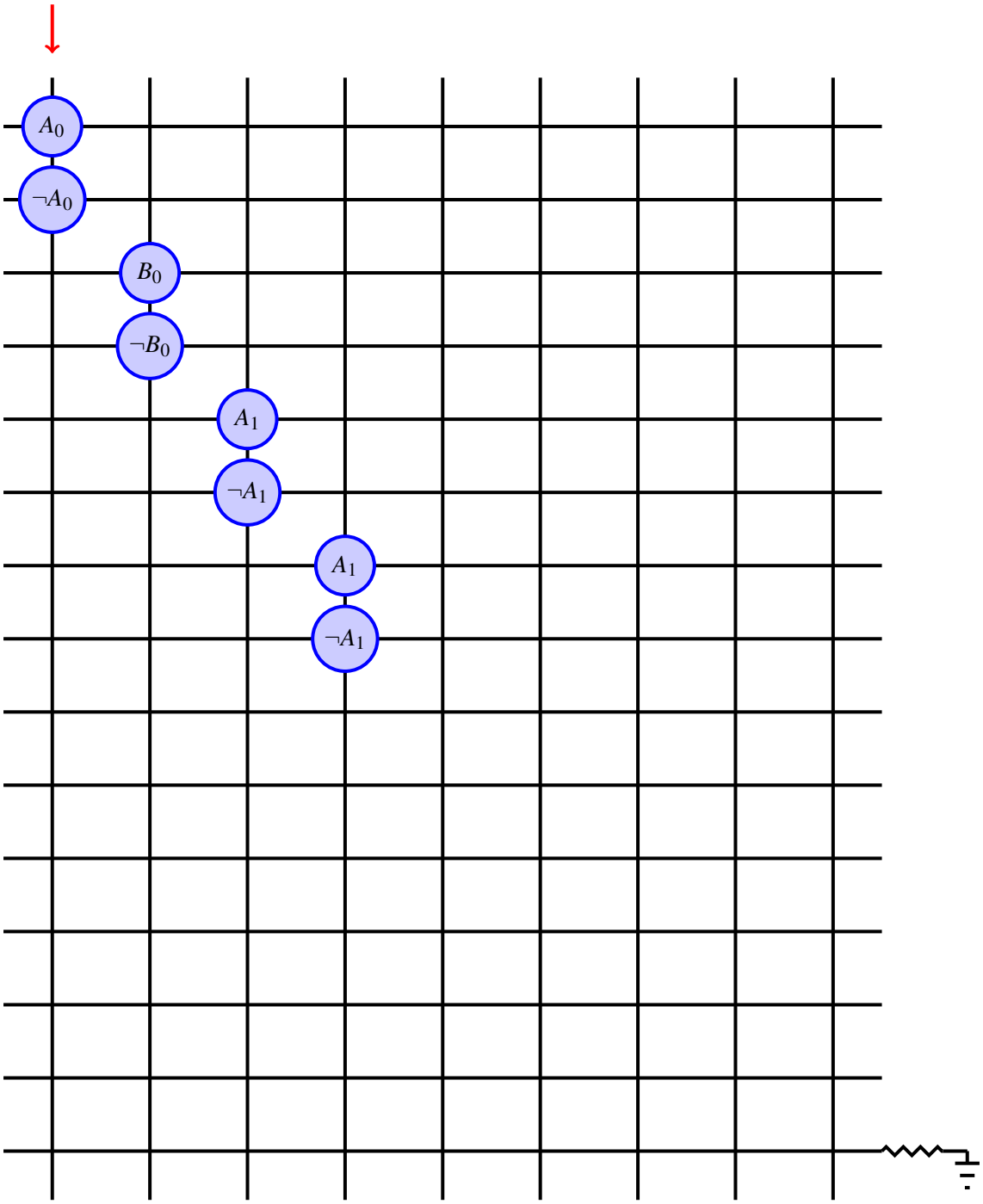
Figure 4.5: Level-by-level mapping of ROBDD to a crossbar



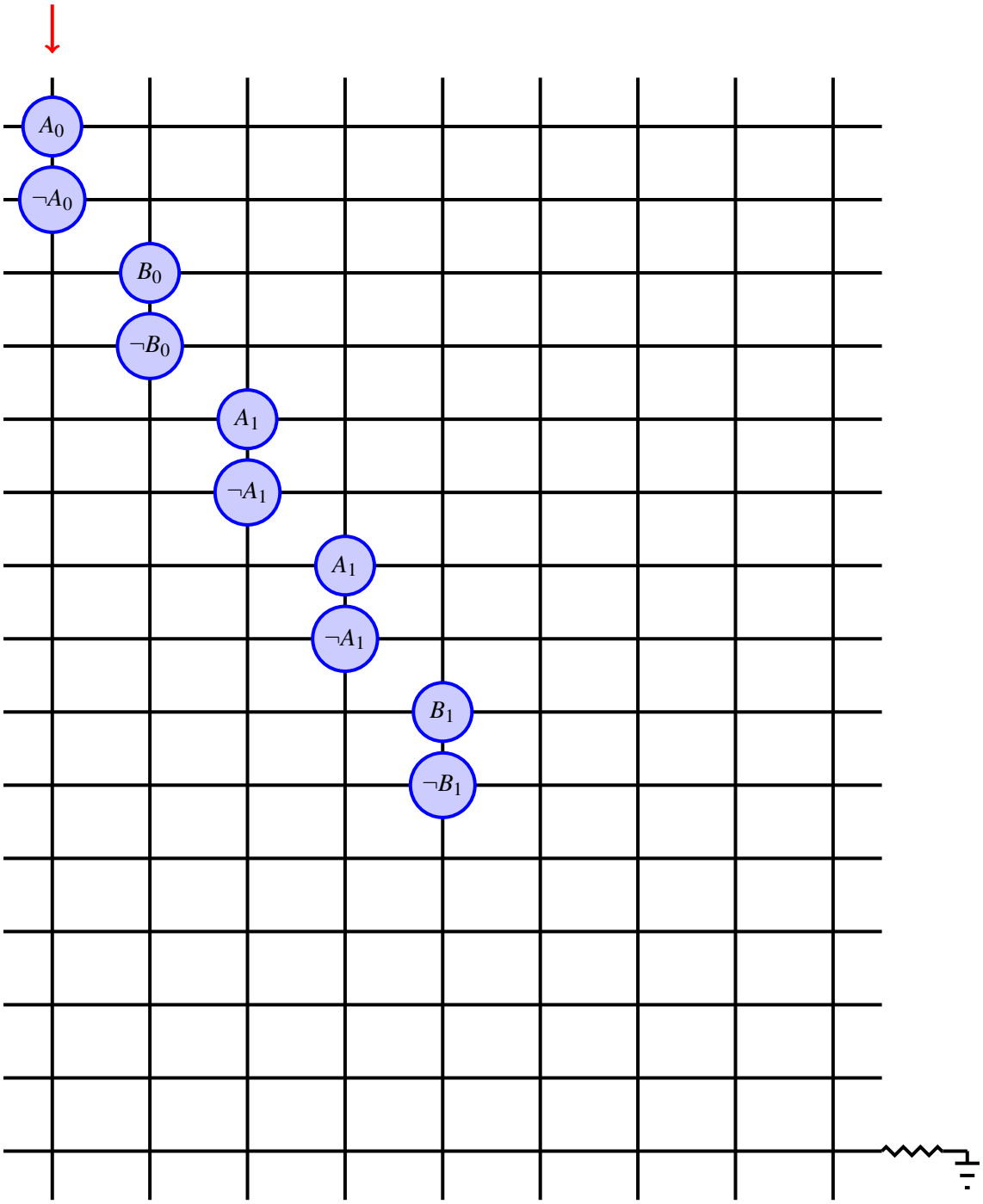
(b) Level 0 mapped



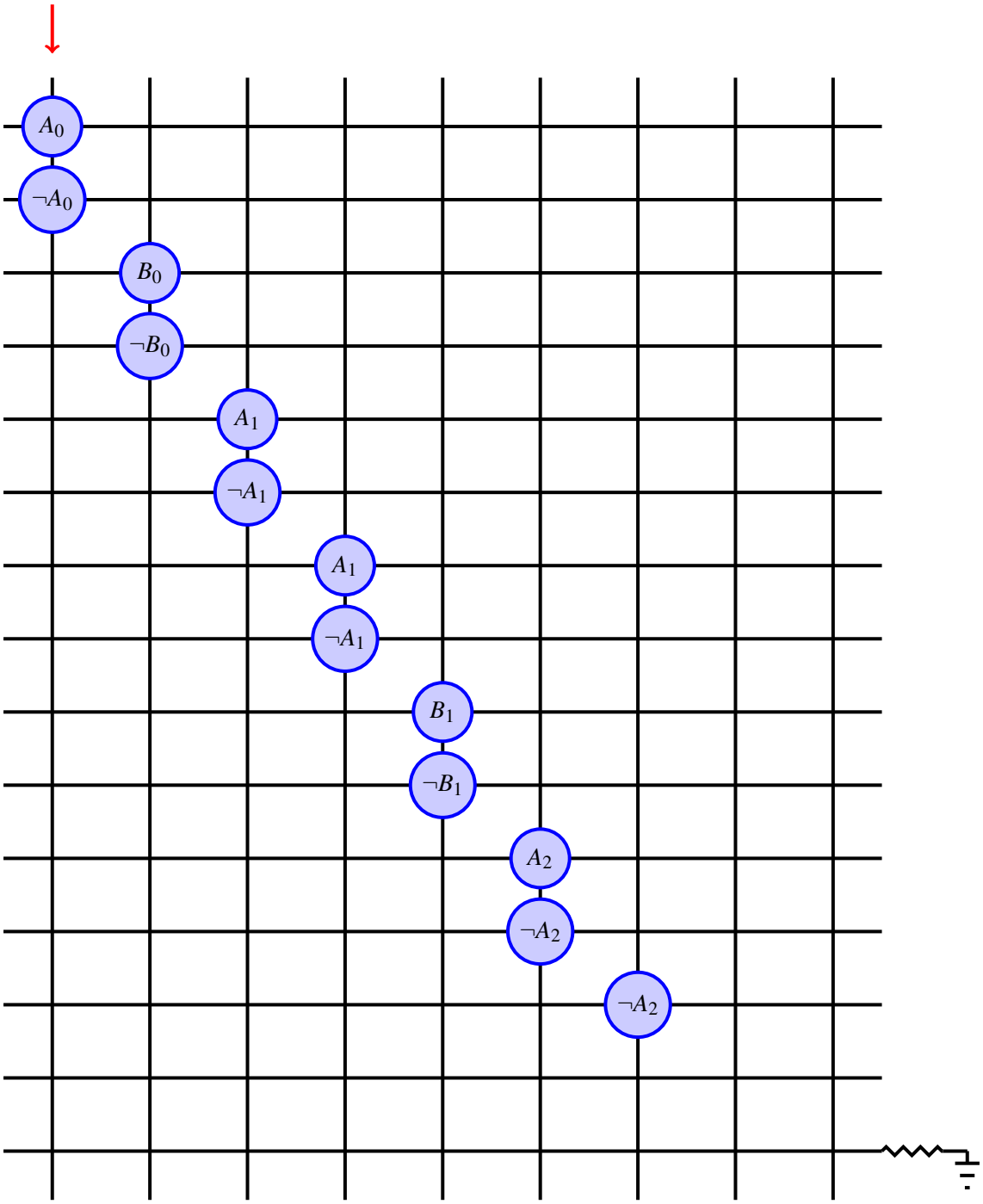
(c) Level 1 mapped



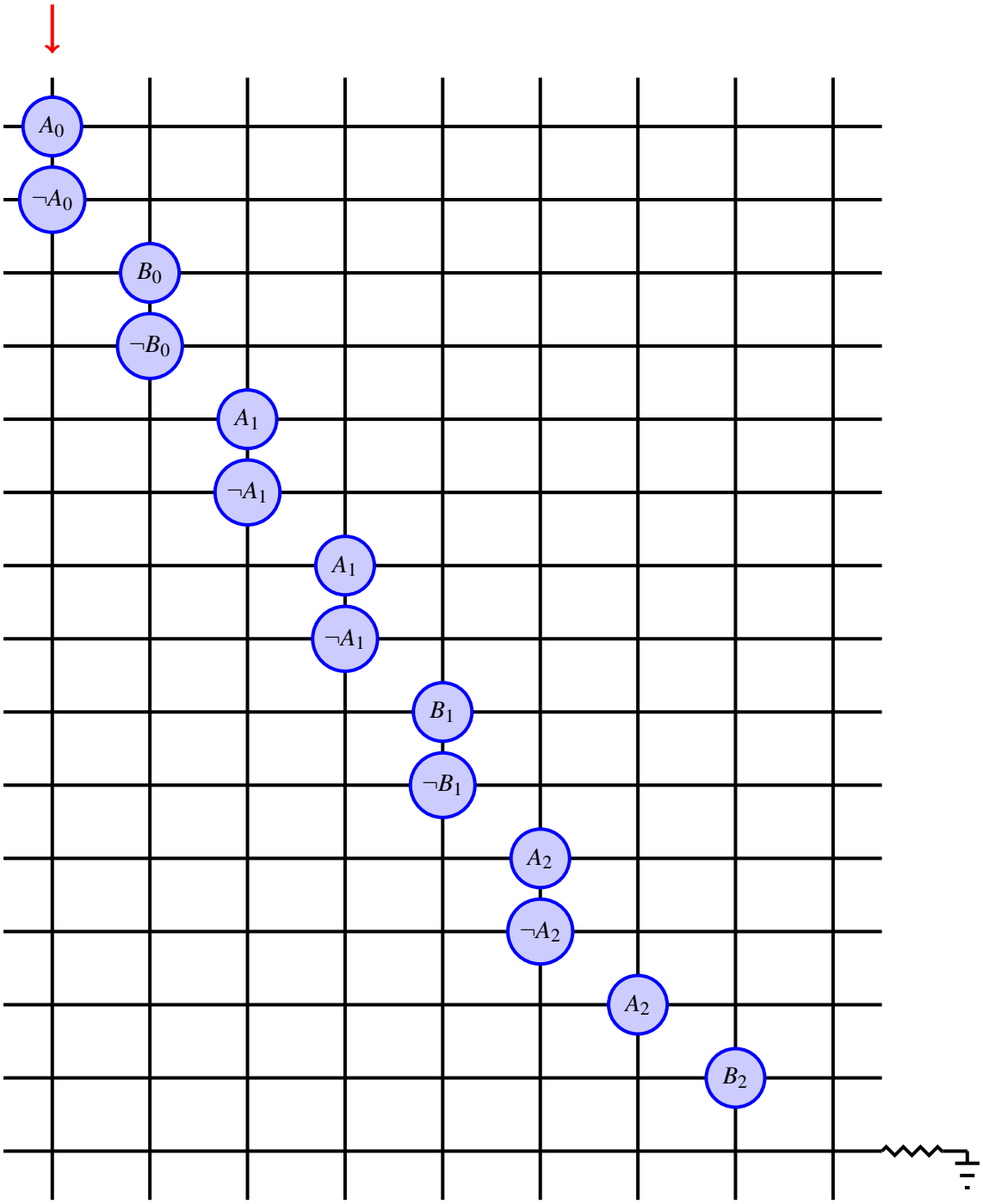
(d) Level 2 mapped



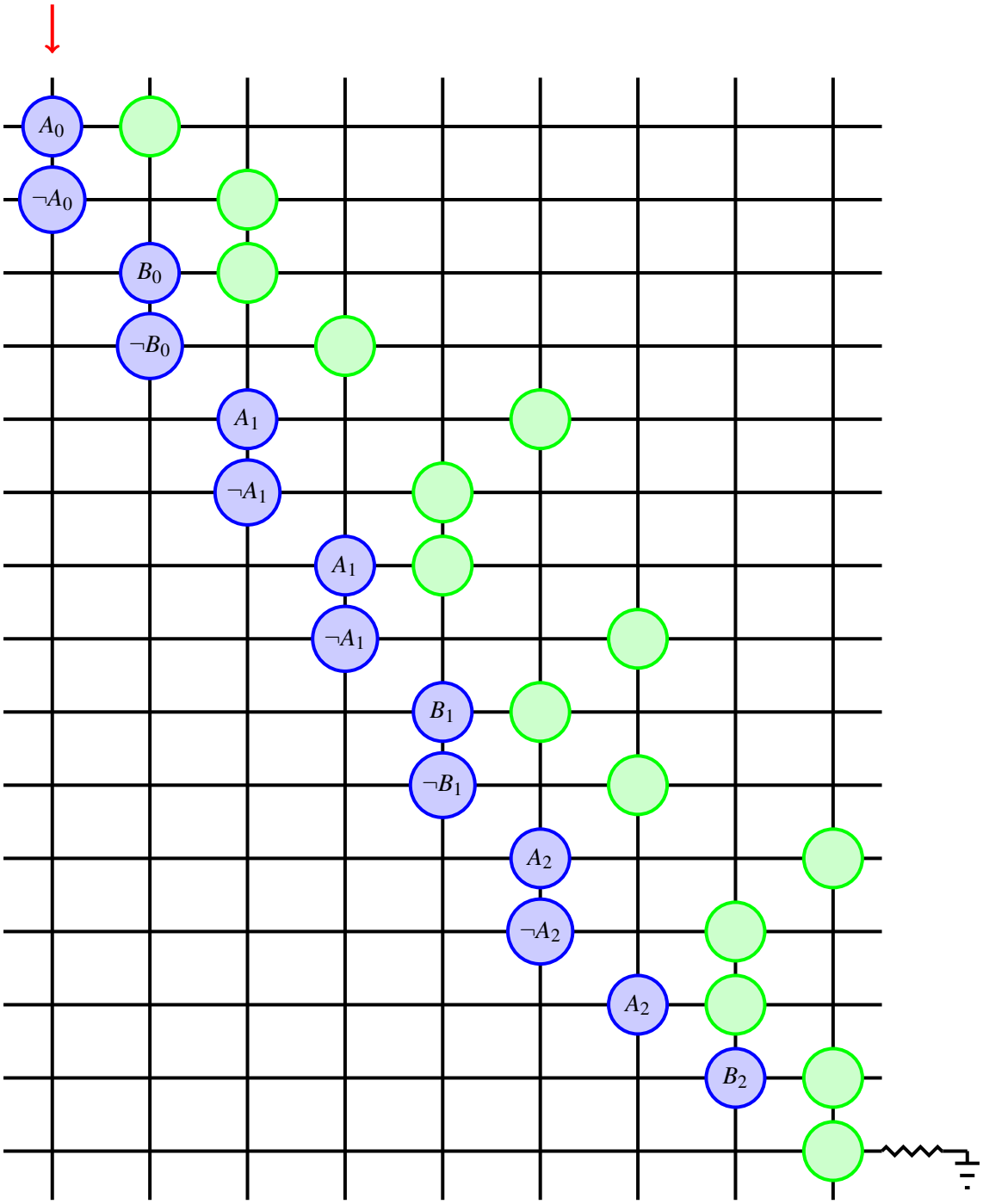
(e) Level 3 mapped



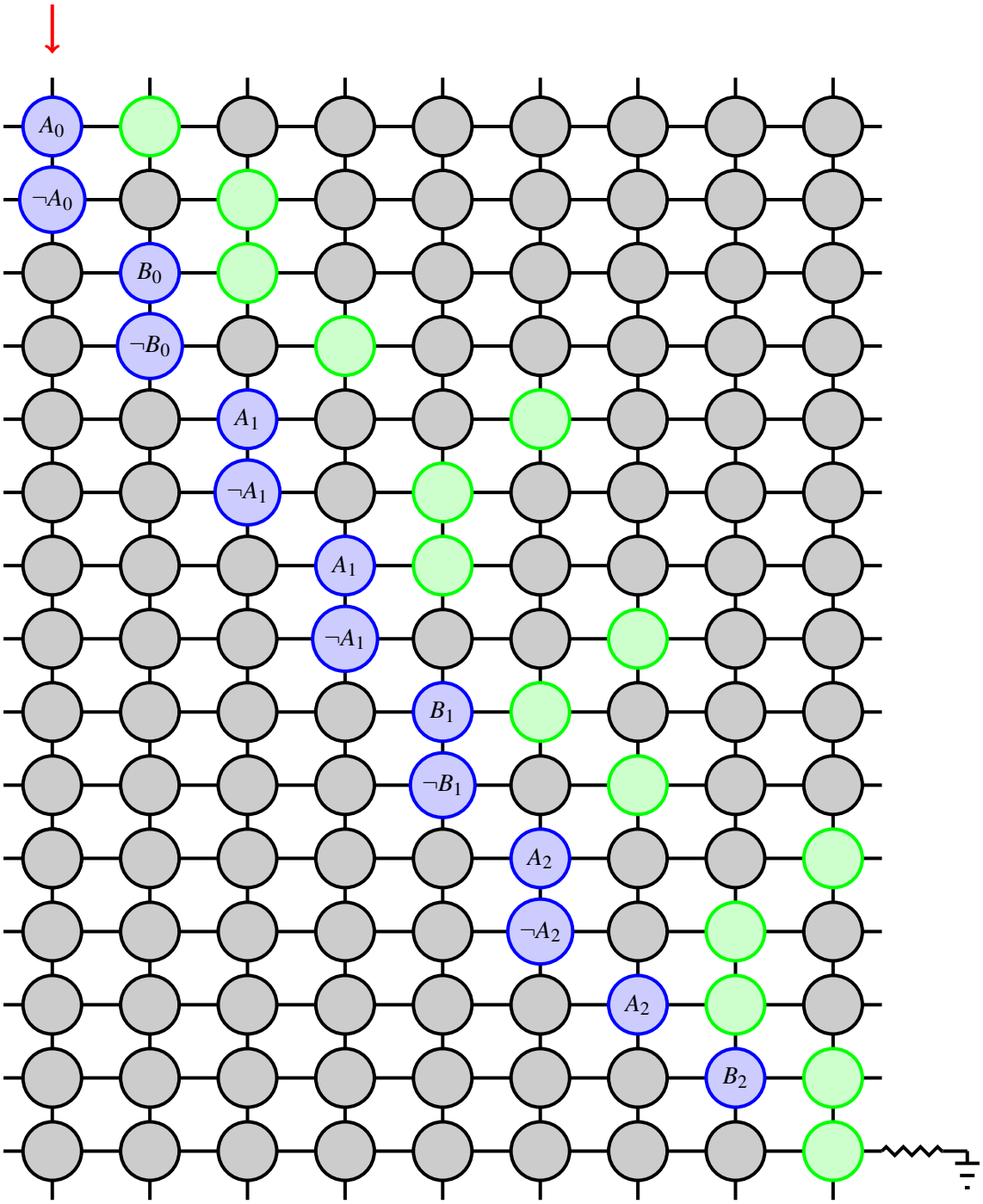
(f) Level 4 mapped



(g) Level 5 mapped



(h) Memristors for edges added



(i) Completely synthesized crossbar

CHAPTER 5: CROSSBAR SYNTHESIS USING MODEL COUNTING

Synthesis of crossbars with bidirectional memristive switches

The utilization of ROBDDs helps us synthesize crossbar circuits for a large number of variables. It is also worth noting that the synthesized crossbars, while functional, are rather inefficient with respect to space constraints. Hence, the synthesis of functional and more compact crossbars makes for a worthwhile pursuit. One of such crossbars is presented in Figure 5.1. The crossbar, despite having only having a size of 4×3 is functional, and computes the most significant bit of 2-bit binary addition.

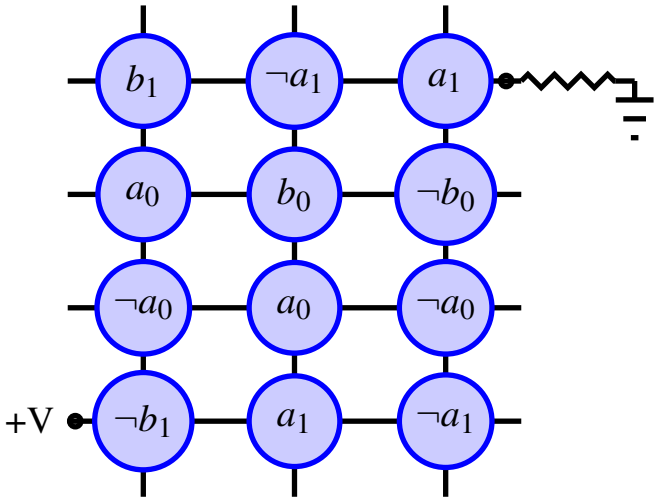


Figure 5.1: Compact crossbar circuit for computing MSB of 2-bit binary addition

The approach for synthesizing compact crossbars driven by simulated annealing [133] is presented in Algorithm 1. In line 1, a crossbar design of size l rows and n columns is chosen randomly. Each memristor in the design is mapped to either *True*, *False*, one of the variables $v_1, v_2, \ldots, v_k$ or one of the negated variables $\neg v_1, \neg v_2, \ldots, \neg v_k$ using a uniform random distribution. For each crossbar design $\mathscr{D}$, we assume that a flow of current is injected into the bottom row (lowermost horizon-

Algorithm 1: Crossbar synthesis algorithm

Input : Target Boolean formula ϕ over variables $\{v_1, v_2, \ldots, v_k\}$
Size of crossbar $\mathbb{C}$: l rows and n columns
Initial temperature for simulated annealing T
Cooling rate c

Output: Crossbar design $\mathscr{D}(\mathbb{M})$ mapping each memristor $m_{ij} \in \mathbb{M}$ to the set $\{True, False, v_1, \ldots, v_k, \neg v_1, \ldots, \neg v_k\}$

1 $\mathscr{D}_1 \leftarrow$ PickRandomCrossbarDesign$(l, n, v_1, \ldots, v_k)$
2 $\mathscr{B}(\mathscr{D}_1) \leftarrow$ BooleanFlow$(\mathscr{D}_1)$
3 $\Delta_1 \leftarrow$ ModelCount$(\mathscr{B}(\mathscr{D}_1) \oplus \phi)$
4 **while** $\Delta_i > 0$ **do**
5 $\quad \mathscr{D}_{i+1} \leftarrow$ PerturbCrossbarDesign$(\mathscr{D}_i, \phi)$
6 $\quad \mathscr{B}(\mathscr{D}_1) \leftarrow$ BooleanFlow$(\mathscr{D}_{i+1})$
7 $\quad \Delta_1 \leftarrow$ ModelCount$(\mathscr{B}(\mathscr{D}_{i+1}) \oplus \phi)$
8 $\quad$ **if** $rand(0,1) < e^{-(\Delta_{i+1} - \Delta_i)/T}$ **then**
9 $\quad\quad i \leftarrow i + 1$
10 $\quad T \leftarrow c \times T$
11 **end**
12 **Return** crossbar design $\mathscr{D}_i$

tal nanowire) of the memristor crossbar. In line 2 of the algorithm, we symbolically compute the Boolean formula representing the values of the memristors under which a flow reaches the topmost nanowire of the crossbar and denote it by BooleanFlow($\mathscr{D}_1$). Let $r_i^{(t)}$ denote the flow value of the column j at time t. At $t = 0$, only the first row has flow, i.e. $r_1^{(0)} = True$ and all other rows and columns are set to $False$. For all $t > 0$, the following transitions are defined for each nanowire, based on the ability of turned-on memristors to create a short-circuit between their horizontal and vertical nanowires:

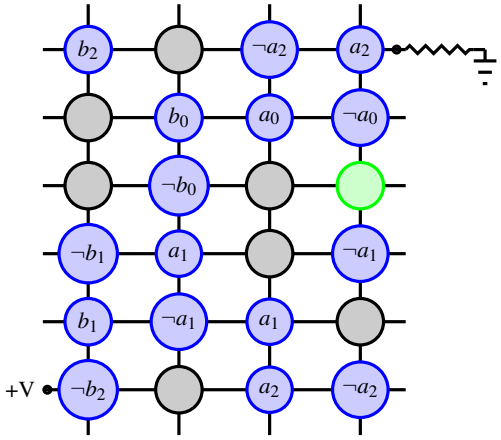
$$\forall i \in \{2, \ldots, n\}, r_i^{(t+1)} \iff (r_i^{(t)} \vee \bigvee_{1 \le j \le n} (m_{ij} \wedge c_j^{(t)}))$$

$$\forall j \in \{1, \ldots, m\}, c_j^{(t+1)} \iff (c_j^{(t)} \vee \bigvee_{1 \le i \le l} (m_{ij} \wedge r_i^{(t)}))$$

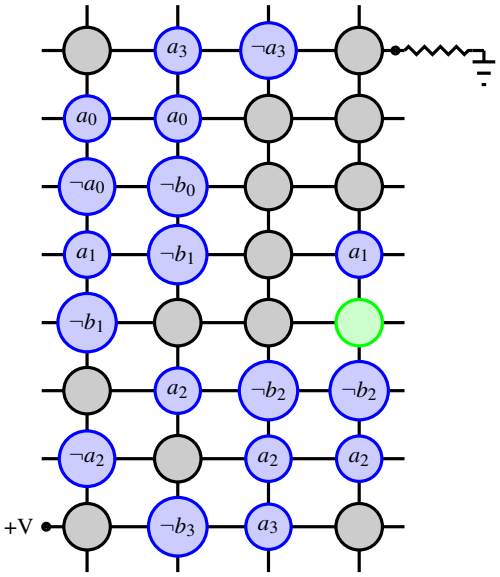
The above transitions of the rows and columns in the crossbar can be represented using Boolean functions and described succintly using Boolean Decision Diagrams (BDDs), And-Inverter-Graphs (AIGs) or other representions.

Line 3 of the algorithm computes the approximate number of satisfiable instances Δ_1 to the Boolean formula corresponding to the symmetric difference of the target Boolean formula ϕ and the formula corresponding to the computation performed by the crossbar design $\mathscr{D}_1$. Several competitive implementations of the approximate model counting algorithms are available – any of these are applicable to our approach, as long as the algorithm produces a count of 0 feasible models only for unsatisfiable formula [134].

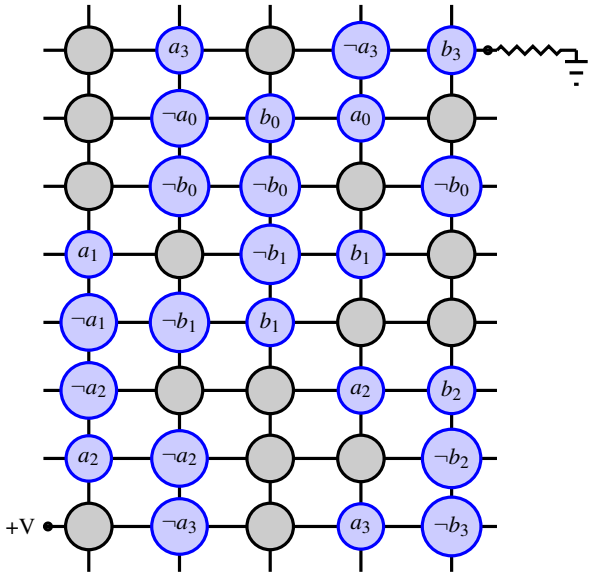
The loop in line 4 through line 12 continues to perturb the crossbar design, evaluate the function that this perturbed design computes and counts the number of satisfiable instances to the Boolean formula and the formula computed by the current crossbar design. Lines 8 through 11 are a succinct description of the simulated annealing algorithm. New crossbar designs are always accepted if they are better than the existing crossbar design. New crossbar designs that are worse than the existing crossbar design are accepted with a probability that is a function of both the quality of the designs and the current temperature of the simulated annealing algorithm. At every iteration of the loop, the temperature of the simulated annealing algorithm is slightly lowered in Line 11 of the pseudocode. At any given point in time, only a single memristor is perturbed. The probability of any given memristor being perturbed is proportional to the number of times the variable corresponding to the memristor occurs in the BDD representation of the symmetric difference between the Boolean formula and design. If a variable does not occur in the symmetric difference, it is not perturbed as the remaining error in the design in not related this variable. When the number of satisfiable instances for the symmetric difference becomes zero, the algorithm stops and reports the synthesized crossbar design. The above approach was presented in [135]. Some interesting compact crossbar designs obtained using the above approach are presented in Figure 5.2.



(a) MSB computation for 3-bit binary addition.



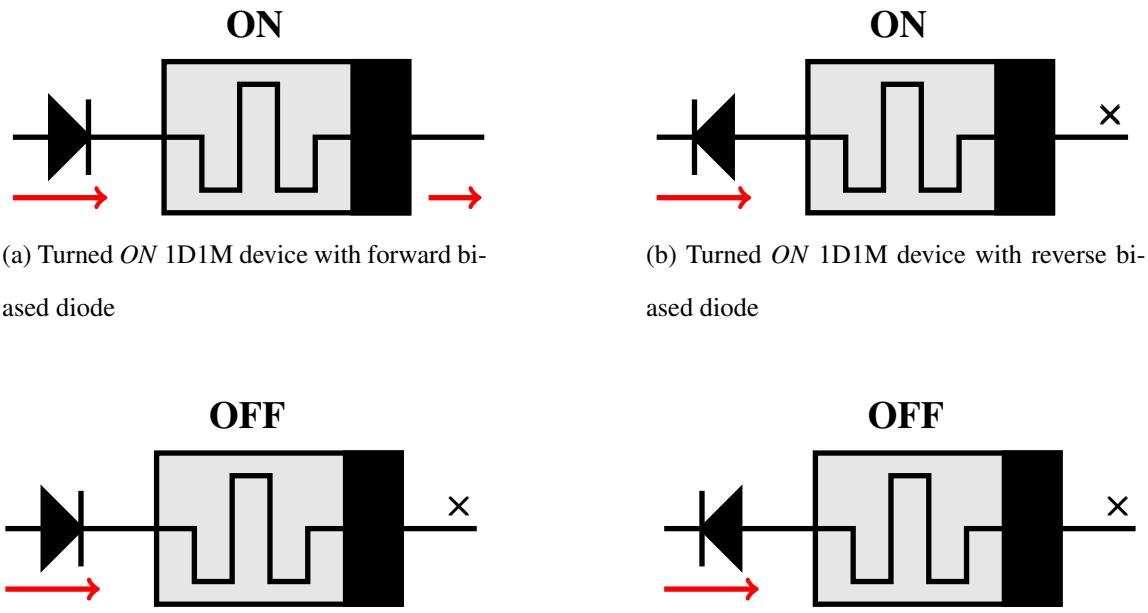
(b) Comparator for 4-bit binary comparison.



(c) MSB computation for 4-bit binary addition.

Figure 5.2: Compact crossbars synthesized using model counting

A single memristor in the *ON* state allows the flow of current through it, regardless of the terminal the current flow is introduced at. While this is an useful property, it can also give rise to paths that are entirely undesired, even within the framework of utilizing sneak paths as a computational mechanism. For this reason, it is necessary to develop and implement memristive switches that allow unidirectional current flow. This is accomplished in a reasonably simple manner in theory, by connecting a single p-n junction diode in series with a memristor – this composite device is termed as a 1D1M device, or a 1D1R device in case of resistive switches which don't necessarily exhibit memristive behavior. A pictorial representation of the fundamental working principles of these devices is provided in Figure 5.3.



(a) Turned *ON* 1D1M device with forward biased diode

(b) Turned *ON* 1D1M device with reverse biased diode

(c) Turned *OFF* 1D1M device with forward biased diode

(d) Turned *OFF* 1D1M device with reverse biased diode

Figure 5.3: Switching states of 1D1M devices

The composite 1D1M devices are also termed as rectifying memristors, due to the presence of diodes. As can be observed from the above figure, a given 1D1M device only allows a current flow if and only if the memristor is turned *ON* and the diode is forward biased. The forward biased diode allows the flow of current through it, as opposed to a reverse biased diode which blocks the flow of current through it. Hence we can see that even if the memristor is turned *ON*, a current flow does not exist at the "output" terminal (terminal other than the one where the current flow is introduced) if the corresponding diode is reverse biased. Similarly, the absence of a current flow at the output terminal is noted when the diode is forward biased but the corresponding memristor is in the OFF state. Quite obviously, there is no current flow at the output terminal if the memristor is in the OFF state, and the corresponding diode is reverse biased as well.

The usage of 1D1M devices empowers us to make an attempt to solve the Feynman Grand Prize challenge. The approach is presented in [136]. The Feynman Grand Prize offered by the Foresight Institute requires the design, construction and demonstration of a nanoscale digital computing device capable of performing 8-bit addition in a cube of edge length 50nm. Assuming that the device is being created using nanoscale memristors of size 3nm [137] and the inter-memristor gap is only 1nm, the 8-bit adder design can still only use crossbars of size 12×12. Such a compact implementation was previously not possible, including the approaches mentioned in the earlier sections of this document. Intuitively, the next step towards the solution would be to utilize a 3-dimensional stack of nanoscale memristor crossbars. For the sake of simplicity in fabrication, it is necessary for the different layers in the stack to have limited and structured interactions. Ideally, such interactions between different layers can be accomplished by using external connections among the layers. Each crossbar layer can operate as a single full adder circuit, and eight such crossbar layers can be stacked on top of each other. Each crossbar layer in the stack needs to implement a full-adder circuit within an area of 50nm$\times$50nm. The 3-D stacked crossbar has a limited number of connections between the different layers. The design of a single crossbar is entirely modular, and computes the sum, the carry-out and the negation of the carry-out for 1-bit Boolean addition. The design of a layer is shown in Figure 5.4.

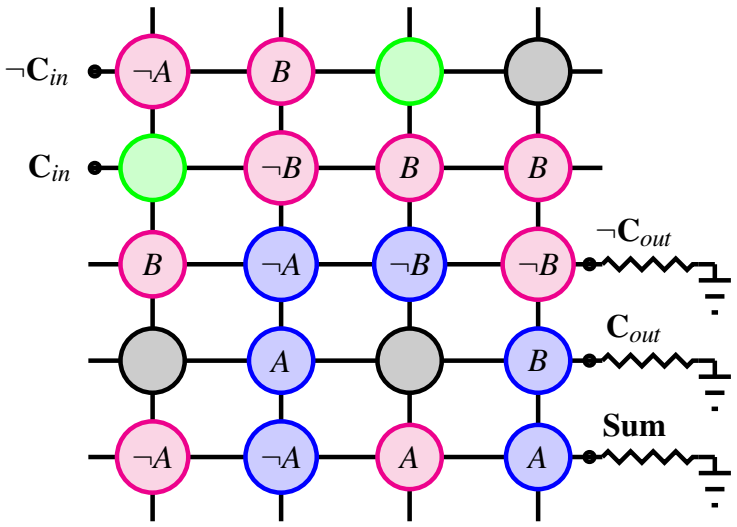
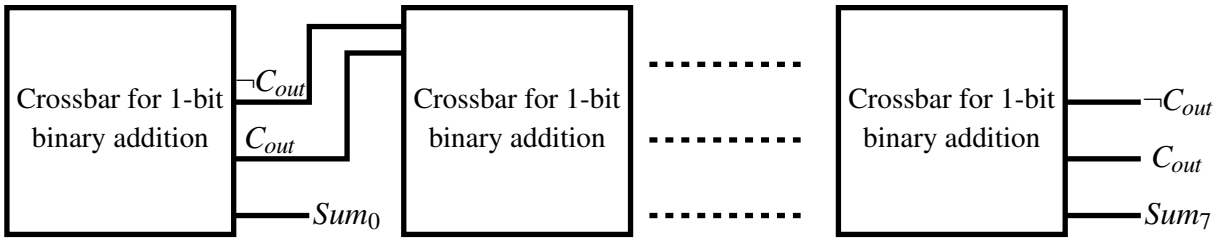


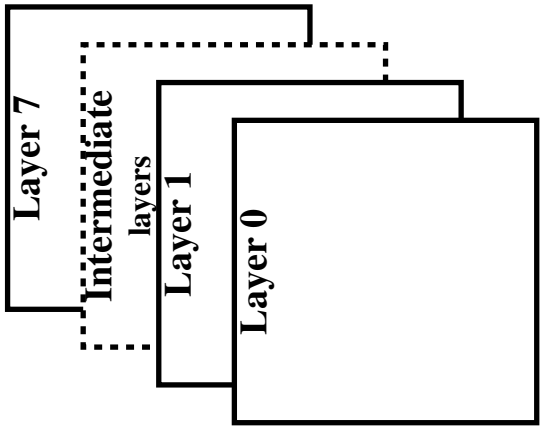
Figure 5.4: A crossbar circuit computing the *Sum*, C_{out} and $\neg C_{out}$ of 1-bit binary addition

The crossbar has five horizontal nanowires (rows) and four vertical nanowires (columns). The top row of the crossbar receives the negation of the carry-in bit, while the second row from the top receives the carry-in bit. The crossbar utilizes two variations of the 1D1M switches. The devices marked in red only allow flow of current from a horizontal nanowire (row) to a vertical nanowire (column). Similarly, the devices marked in blue only allow flow of current from a vertical nanowire (column) to a horizontal nanowire (row). Switches that are constantly in the *ON* state and *OFF* state are marked in green and gray, respectively. The outputs are obtained from the bottom row, the second row from the bottom and the third row. If the *Sum* bit is true, a flow is observed in the bottom row. If the C_{out} bit is true, then a flow is observed at the second row from the bottom. If the C_{out} bit is false, then no flow is observed the second row from the bottom, but a flow is observed at the third row.

A compact 8-bit adder is constructed by stacking eight identical crossbar layers on top of each other. The pattern of external connections and stacking of the crossbars presented in Figure 5.5. The inputs to the crossbars should be loaded into the memristors according to the design in Figure 5.4. The carry-in bit for the first crossbar should be set to 0 and the negation of the carry-in bit should be set to 1.



(a) Linear representation of 3D stacking and external connections



(b) 3D stacking of identical crossbar layers

Figure 5.5: 3D stacking of crossbar circuits and external connections between them

For all the other carry-in bits, the nanowire corresponding to the carry-out bit of the previous full-adder module should be fed into the carry-in bit of the next full-adder circuit. Similarly, the negation of the carry-out bit should be fed into the negation of the carry-in bit of the next full-adder module. The eight crossbars will produce the eight sum-bits of the 8-bit adder on their lowest nanowires.

It is worth noting that compactness of the synthesized designs is driven largely by the effective utilization of 1D1M devices. Modularity requires our flow-based designs to receive two input flows: the carry-in bit and its negation. In purely memristive crossbar consisting only of bi-directional switches, these two flows get mixed-up and produce erroneous results in absence of unidirectional 1D1M devices.

CHAPTER 6: USING FLOW-BASED CROSSBARS FOR IN-MEMORY EXECUTION OF COMPUTE KERNELS

Till now, the predominant computer architectures have all been transistor based. Hence, the programming paradigm has also evolved to mainly accommodate the usage of transistor-based computers. It is worth noting that memristor based crossbars offer a computational fabric that is fundamentally different from transistor-based systems. Among other advantages, the crossbars offer higher spacial density, faster switching times and greater power efficiency over conventional transistor-based systems. In addition to that, memristors and resistive switches also provide non-volatility. Even with such superior features, care must be taken when shifting from the old systems to the new architectures, from a programming standpoint. In this section and in [138], we take a look at the possible future of programming crossbars in a sustainable manner. As has been discussed in the previous sections, flow-based crossbar computing allows data to be loaded onto a memristor crossbar in a well-designed structural pattern such that the flow of current through the crossbar can be used to perform Boolean computations. Such an in-memory memristive computing approach may be useful for applications where the same computation needs to be performed on evolving data sets. However, it a non-trivial task for an application developer to map even the simplest programs onto such nanoscale memristor crossbars.

Flow-based computing using nanoscale memristor crossbars is particularly suited for single-instruction multiple-data (SIMD) parallelism. The application developer can be expected to write *kernels* in a subset of the C language. Such a kernel should be algorithmically mapped onto a memristor crossbar – thereby relieving the programmer of the need to learn a new programming model. Several interesting applications in computer vision, linear algebra and machine learning can benefit from such a compilation of restricted C programs to nanoscale memristor crossbars. Our approach uses thr LLVM compilation framework [139] to transform the compute kernel code in the C programming language into the LLVM intermediate representation. The LLVM IR is then transformed into

a Boolean Decision Diagram (BDD) and the decision diagram is then mapped onto a nanoscale memristor crossbar. The parallel Xyces electronic simulation software is then utilized to test the correctness of the designed nanoscale memristor crossbars.

Compute Kernel

The compute kernel has a structure representing the inputs that it needs to use. Each input can be an integer whose bit width is specified during compilation. In the example code provided below, the kernel for edge detection has two inputs: p1 representing one pixel value and p2 representing a pixel value next to p1. The kernel also has an output structure that has at least one output variable. Each output variable is an integer whose bit width is again specified during compilation. In the given example, the output structure has a single integer output o.

```
// Edge detection kernel
struct Input {int p1; int p2; } input;
struct Output {int o; } output;

void compute()
{
    output.o = input.p2 - input.p1;
}
```

The core of the kernel is the *compute*() function that uses the syntax of the C language to define how the input variables must be manipulated to produce the output variable. The C code inside the kernel should not have loops that cannot be statically unrolled.

From C Kernel to Intermediate Representation

The C kernel code is compiled using the LLVM compiler to an intermediate representation. The structure of the kernel definition makes it easier to obtain the core computations being performed by the kernel code. The compiled intermediate representation for the running example is provided at the top of the next page. The LLVM intermediate at representation provides a simple linear list of arithmetic and logical operations that can then be transformed to Boolean Decision Diagrams.

```
define void @compute()  {
%1 = load i32 , i32* getelementptr inbounds (%struct.Input ,
    %struct.Input* @input , i64 0 , i32 0 ), align 4, !tbaa !1
%2 = load i32 , i32* getelementptr inbounds (%struct.Input ,
    %struct.Input* @input , i64 0 , i32 0 ), align 4, !tbaa !6
%3 = sub nsw i32 %1, %2
store i32 %3, i32* getelementptr inbounds (%struct.Output,
%struct.Output* @output, i64 0, i32 0), align4, !tbaa !7
ret void
}
```

Intermediate Representation to Boolean Decision Diagram

Boolean Decision Diagram packages like BuDDy provide robust algorithms for mapping logical and linear arithmetic operations into Boolean Decision Diagrams. The LLVM intermediate representation obtained from the computational kernel makes direct references to arithmetic and logical computations on its operands and enables as one-to-one mapping to Boolean Decision Diagram operations.

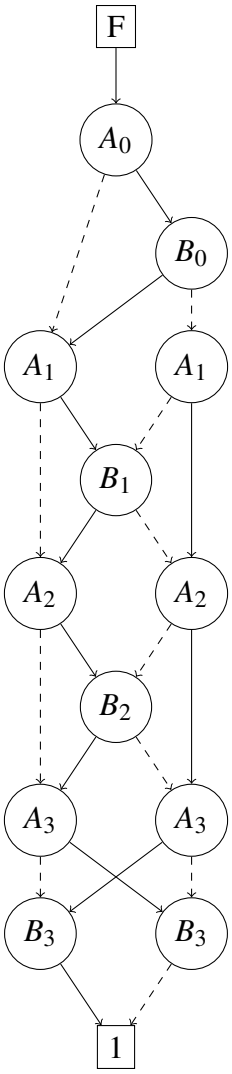


Figure 6.1: ROBDD for the MSB of 4-bit binary subtraction

In our running example, the LLVM intermediate representation states "%3 = sub nw i32 %1 %2" indicating that a subtraction operation should be performed using the operands in the first two positions on the input structure and the result returned as the first element of the output structure. This is readily achieved in our approach using the bit-vector subtraction operation in the BuDDy package. Alternate layers of the BDD interleave the Boolean variable in Figure 6.1, which corresponds to an efficient ordering of the operand bits.

Non-linear arithmetic operations like multiplication lead to an explosion in the size of the BDD as the number of bits in the input increases. Hence, the approach based on BDDs is not really applicable to such non-linear computations. Once the ROBDD is obtained, it can be mapped to a crossbar using the methodology described in section 3. Once the crossbars are synthesized, they are validated through simulation using Xyces. This is discussed in the following section.

Illustrative Example

The concepts described in this section are further explored in order to implement a simple edge detection framework in RBG images. The inputs to the kernel include RGB values of two pixels and the output is 4 less than the difference of the sum of the RGB values of two pixels. Intuitively, if this output is less than 0, there is no edge at the location of a given pixel. The C code for this example is provided below.

```
// RGB edge detection kernel code
struct Input {int r1; int g1; int b1; int r2; int g2; int b2;
    int r3; int g3; int b3;} input;
struct Output {int o; } output;

void compute()
{
    int avg1, avg2, diff;
    avg1 = input.r1+input.g1+input.b1;
    avg2 = input.r2+input.g2+input.b2;
    diff = avg2-avg1;
    output.o = diff-4;
}
```

Using the above-mentioned approach, the code is transformed into its equivalent LLVM intermediate representation, as given below.

```
%7 = sub i32 -4, %1
%8 = sub i32 %7, %2
%9 = sub i32 %8, %3
%10 = add i32 %9, %4
%11 = add i32 %10, %5
%12 = add i32 %11, %6
```

This intermediate representation is then transformed into a BDD, which is mapped onto a memristive crossbar, which is then validated through electronic simulation with the Xyces parallel simulator.

CHAPTER 7: EXPERIMENTAL RESULTS

ROBDD based synthesis

Synthesis of crossbar circuits is an interesting topic of research in computer science. To this end, there exist some approaches which attempt to synthesize efficient crossbar circuits. Some of these approaches also utilize BDDs [140], or they might use similar graph-based data structures like Majority Inverter Graphs (MIGs) which utilize both implication (MIG-IMP) and majority (MIG-IMP) functions, as shown in [141]. Similarly, we also compare the relevant results from our own ROBDD-based work [132] (as described in Section 3) with results obtained from existing solutions both NNF-based [116] and SMT-based [118, 119], all of which use sneak-paths as a computational mechanism – this is presented in Table 7.1.

Table 7.1: Size comparison of crossbars for n-bit Boolean addition

No. of bits/input	NNF-based	SMT-based	ROBDD-based
8	$10^4 \times 10^4$	Time Out	32×17
16	$10^6 \times 10^6$	Time Out	64×33
32	$10^{11} \times 10^{11}$	Time Out	128×65
64	$10^{21} \times 10^{21}$	Time Out	256×129
128	$10^{41} \times 10^{41}$	Time Out	512×257

As can be seen from the above table, the NNF-based method produces very large crossbars and the SMT-based methods fail to produces any results for higher values of n, while our ROBDD-based method manages to produce reasonably succinct crossbars even for larger formulas. Next, let us compare the performance of the crossbars produced by the previously mentioned methods and our

ROBDD-based method. It is worth pointing out that the BDD-based method is provided in [140] and is *very* different from our ROBDD-based approach. In Table 7.2, the last column represents the speedup and in Table 7.3, the last column represents the power consumption ratio. For a given memristor, the switching delay is the time is takes for the memristor to switch from an *ON* state to an OFF state and vice-versa. Similarly, the power is power consumed by one memristor to switch from the *ON* state to the OFF state. The values for these two parameters – switching delay and switching power consumption, are adopted from the devices described in [142].

Table 7.2: Delay (in picoseconds) to compute the MSB of n-bit binary addition

No. of bits/input	BDD	MIG-IMP	MIG-MAJ	ROBDD-based	Improvement
16	16,320	27,200	8,160	5,369	151.9%
32	32,640	54,400	16,320	10,823	150.7%
64	65,280	108,800	32,640	21,729	150.2%
128	130,560	217,600	65,280	43,543	149.9%

Table 7.3: Power consumption (in μW) to compute the MSB of n-bit binary addition

No. of bits/input	BDD	MIG-IMP	MIG-MAJ	ROBDD-based	Improvement
16	5,760	9,600	2,880	1,901	151.5%
32	11,520	19,200	5,760	3,832	150.3%
64	23,040	38,400	11,520	7,693	149.7%
128	46,080	76,800	23,040	15,415	149.5%

In addition to the adder circuits, the approach is also used to synthesize crossbar circuits for selected benchmarks from [143]. The results are provided in Table 7.4 and Table 7.5.

Table 7.4: Delay (in picoseconds) for selected benchmarks

No. of bits/input	BDD	MIG-IMP	MIG-MAJ	ROBDD-based	Improvement
5xp1	6,205	8,415	3,060	1,109	275.9%
9sym_d	5,270	14,875	5,100	2,898	200.1%
misex3	15,725	14,025	5,695	5,369	106.1%
sym10_d	5,950	15,895	6,120	4,192	145.9%
clip	7,585	9,350	3,400	1,706	199.3%

Table 7.5: Power consumption (in μW) for selected benchmarks

No. of bits/input	BDD	MIG-IMP	MIG-MAJ	ROBDD-based	Improvement
5xp1	2,190	2,970	1,080	393	274.8%
9sym_d	1,860	5,250	1,800	1,027	175.3%
misex3	5,550	4,950	2,010	1,901	105.7%
sym10_d	2,100	5,610	2,160	1,479	146.0%
clip	2,670	3,300	1,200	605	198.3%

Furthermore, it is of utmost importance the output signals for *True* and *False* are not ambiguous. In the following Table 7.6, we can observe from the simulation results there is no overlap in the voltage ranges for the *True* and *False* signals. The minimum value of a *True* output (in Volts) is at least 10 times larger than the corresponding maximum value of a *False* output (in Volts). Hence, it is practical to distinguish false values from true values unambiguously.

Table 7.6: Simulation results for n-bit Boolean addition

No. of bits/input	Maximum value of false signal	Minimum value of true signal
2	4×10^{-5}	0.1
4	8×10^{-5}	0.056
8	16×10^{-5}	0.031
16	31×10^{-5}	0.017
32	22×10^{-5}	0.009
64	20×10^{-5}	0.005
128	17×10^{-5}	0.002

Model counting based synthesis

The attempt to synthesize crossbar circuits based on a model counting approach yields memristor crossbars that are more compact in comparison to the crossbars produced by the ROBDD-based approach. The comparison with respect to size are provided in Table 7.7, for computing the MSB of n-bit binary addition. The last column in the table represents the percentage decrease in the crossbar size with respect to the prior best best method of crossbar synthesis.

Table 7.7: Comparison of crossbars sizes for n-bit binary addition

No. of bits/input	NNF-based	SMT-based	ROBDD-based	Model counting	Size decrease
2	64×64	Time Out	8×5	4×4	250%
3	184×184	Time Out	15×9	6×4	562.5%
4	472×472	Time Out	21×12	8×5	630%

As expected, the compactness in size of the area-optimized crossbars is accompanied by faster computation times and higher power efficiency, as shown in Table 7.8 and Table 7.9, respectively. The last column in Table 7.8 represents the speedup achieved through model counting with respect to the prior best method. Similarly, the last column in Table 7.9 represents the power consumption ratio of crossbars produced through model counting with respect to the prior best method. All of the above crossbars compute the MSB of n-bit binary addition.

Table 7.8: Delay (in picoseconds) for n-bit binary addition

No. of bits/input	ROBDD	MIG-IMP	MIG-MAJ	Model counting	Speedup
2	425	3400	1020	340	125%
3	765	5100	1530	340	225%
4	1020	6800	2040	425	240%

Table 7.9: Power consumption (in μW) for n-bit binary addition

No. of bits/input	ROBDD	MIG-IMP	MIG-MAJ	Model counting	Speedup
2	240	1200	360	300	80%
3	420	1800	540	390	107.7%
4	600	2400	720	720	83.3%

It can be easily observed that there is only marginal advantage in terms of power consumption when using the model counting approach. It is also worth noting that this approach is unable to synthesize the requisite crossbars for higher values of n in case of n-bit inputs. The simulations for this approach and the ROBDD-based approach were all conducted using ngSPICE-26 [144].

The circuits developed for the in-memory execution of the simple edge detection kernels are executed in the Xyces simulator [145] simulator using its built-in memristor device model. In Table 7.10, the first column represents the index of the output bit, the second column represents the expected logical state of the output bit, the third column represents the expected flow state at the output nanowire and the final column represents the measured voltage (in Volts) at the output nanowire. The inputs provided in this case are: p1=1011 and p2=1000.

Table 7.10: Simulation results for crossbars implementing a simple edge detection kernel

Output bit	Logical state	Flow state	Output voltage (Volts)
0	1	Flow	1.9×10^{-3}
1	0	No-flow	1.8×10^{-11}
2	1	Flow	4.3×10^{-4}
3	1	Flow	1.9×10^{-4}

Hence, we can see that the crossbar does indeed perform correctly. A similar experiment is performed with the crossbar computing the edge detection kernel for RGB pixels. In addition to the expected and observed outputs, the crossbar sizes for computing the output bits is also provided in Table REF HERE. The inputs provided in this case are as follows: b1 = 0001, b2=0101, g1 = 1000, g2 = 1001, r1 = 1001 and r2 = 1101.

Table 7.11: Simulation results for crossbars implementing RGB edge detection

Output bit	Logical state	Flow state	Output voltage (Volts)	Crossbar size
0	1	Flow	3.3×10^{-4}	21×12
1	0	No-flow	2.1×10^{-12}	55×29
2	1	Flow	3.3×10^{-6}	103×53
3	0	No-flow	1.9×10^{-14}	171×87

As can be seen above, the crossbars perform as expected.

CHAPTER 8: CONCLUSION

Emerging high-density non-volatile memory technologies such as memristive RAMs provide a new opportunity for coupling computation with memory on the same fabric. Earlier approaches to computing have suffered from a number of different drawbacks. In this document, new research is presented, which attempts to mitigate the problems inherent in the current methods to automatically synthesize sneak-path based memristor crossbars. The proposed approaches and their corresponding results accomplish the following goals:

a. The size of our crossbar circuits can be exponentially more succinct than the size of those designed using structural induction on negation normal forms.

b. Crossbar circuits have been designed for Boolean formulae that are two orders of magnitude larger than those designed using SMT solvers.

c. It has been shown that a given Boolean formula can be computed using crossbars at most linear in the size of the ROBDD representation of the formula.

d. Compact crossbars have been designed for interesting examples of Boolean formulae such as 4-bit adder and 4-bit comparator.

e. The compact crossbars are found be up to 3 time faster than previous approaches and requires up to 6.3 times less area than the same. The corresponding energy performance is at least as good as contemporary approaches based on other forms of abstraction.

f. Compute kernels rich in logical and linear arithmetic computations have been automatically compiled into flow-based crossbar computing designs.

g. The LLVM compilation framework and the BuDDy BDD package has been used for transformation of well-designed compute kernels into in-memory computing accelerators, along with parallel simulation of the synthesized designs using Xyces.

h. All of the crossbar designs produced have been been validated by logical enumeration, and simulated using well-studied memristor models and SPICE software.

In conclusion, novel electronic design automation approaches have been demonstrated for automated synthesis of sneak-path based crossbar circuits using memristive switching devices. The work spans computation of n-bit Boolean formulae with linearly growing circuits, 4-bit Boolean formulae with compact circuits, 8-bit Boolean addition with stacked 2-D crossbars, and compilation of a subset of C programs into crossbar circuits.

CHAPTER 9: FUTURE WORK

The drawbacks of the work presented in this dissertation and the tasks that can be undertaken in order to tackle them can be summarized as follows:

- For certain classes of Boolean formulas, ROBDDs grow exponentially. In such cases, application of the methods we have presented in [132].
- Mapping of more succinct data structures like and-inverter graphs, majority-inverter graphs, xor-inverter graphs and free binary decision diagram, to crossbar circuits is a future area of research.
- Model counting can be limited in its efficacy due to the large number of evaluations.
- Approximation algorithms, especially ones that can leverage the power of existing SIMD microprocessors, can augment the search process for compact crossbar designs. This remains an exciting area for future research.
- Much work needs to done on the development of a testing and debugging framework for compiling C programs to crossbar circuits.
- Synthesis of large fault-tolerant crossbars via intelligent restructuring of ROBDDs is an interesting direction of research.
- Harnessing the capabilities of artificial intelligence methods like deep learning for automated synthesis of crossbar circuits is a very exciting avenue for future research.
- Intuitively, we can speculate that computation of ternary and m-valued logic operations on crossbar circuits should be possible by exploiting the state property of memristors and other resistive memory devices. Currently, there does not exist a reliable body of work supporting this idea, which makes it a fertile area for future work.

LIST OF REFERENCES

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[2] G. E. Moore *et al.*, "Progress in digital integrated electronics," in *Electron Devices Meeting*, vol. 21, pp. 11–13, 1975.

[3] G. Baccarani, M. R. Wordeman, and R. H. Dennard, "Generalized scaling theory and its application to a $1/4$ micrometer mosfet design," *IEEE Transactions on Electron Devices*, vol. 31, no. 4, pp. 452–462, 1984.

[4] E. Mollick, "Establishing moore's law," *IEEE Annals of the History of Computing*, vol. 28, no. 3, pp. 62–75, 2006.

[5] X. Huang, W.-C. Lee, C. Kuo, D. Hisamoto, L. Chang, J. Kedzierski, E. Anderson, H. Takeuchi, Y.-K. Choi, K. Asano, *et al.*, "Sub-50 nm p-channel finfet," *IEEE Transactions on Electron Devices*, vol. 48, no. 5, pp. 880–886, 2001.

[6] A. W. Topol, D. La Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, *et al.*, "Three-dimensional integrated circuits," *IBM Journal of Research and Development*, vol. 50, no. 4.5, pp. 491–506, 2006.

[7] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[8] K. J. Kuhn, "Moore's law past 32nm: Future challenges in device scaling," in *Computational Electronics, 2009. IWCE'09. 13th International Workshop on*, pp. 1–6, IEEE, 2009.

[9] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, no. 3, pp. 144–149, 2002.

[10] F. Stern and W. Howard, "Properties of semiconductor surface inversion layers in the electric quantum limit," *Physical Review*, vol. 163, no. 3, p. 816, 1967.

[11] J. R. Powell, "The quantum limit to moore's law," *Proceedings of the IEEE*, vol. 96, no. 8, pp. 1247–1248, 2008.

[12] J. Von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[13] A. W. Burks, H. H. Goldstine, and J. Von Neumann, "Preliminary discussion of the logical design of an electronic computer instrument," 1946.

[14] J. Von Neumann, "The principles of large-scale computing machines," *Annals of the History of Computing*, vol. 3, no. 3, pp. 263–273, 1981.

[15] J. Backus, "Can programming be liberated from the von neumann style?: a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[16] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," *IEEE micro*, vol. 17, no. 2, pp. 34–44, 1997.

[17] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[18] A. J. Smith, "Cache memories," *ACM Computing Surveys (CSUR)*, vol. 14, no. 3, pp. 473–530, 1982.

[19] D. M. Tullsen and S. J. Eggers, "Limitations of cache prefetching on a bus-based multiprocessor," in *ACM SIGARCH Computer Architecture News*, vol. 21, pp. 278–288, ACM, 1993.

[20] A. Kagi, J. R. Goodman, and D. Burger, "Memory bandwidth limitations of future microprocessors," in *Computer Architecture, 1996 23rd Annual International Symposium on*, pp. 78–78, IEEE, 1996.

[21] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," in *ACM SIGARCH Computer Architecture News*, vol. 3, pp. 126–132, ACM, 1975.

[22] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "Computation-in-memory based parallel adder," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pp. 57–62, IEEE, 2015.

[23] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE micro*, vol. 27, no. 5, pp. 15–31, 2007.

[24] J. Yu, R. Nane, A. Haron, S. Hamdioui, H. Corporaal, and K. Bertels, "Skeleton-based design and simulation flow for computation-in-memory architectures," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*, pp. 165–170, IEEE, 2016.

[25] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 336–348, IEEE, 2015.

[26] T. L. Sterling and H. P. Zima, "Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing," in *Supercomputing, ACM/IEEE 2002 Conference*, pp. 48–48, IEEE, 2002.

[27] Q. Zhu, K. Vaidyanathan, O. Shacham, M. Horowitz, L. Pileggi, and F. Franchetti, "Design automation framework for application-specific logic-in-memory blocks," in *Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on*, pp. 125–132, IEEE, 2012.

[28] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, ACM, 1967.

[29] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 148–151, 2015.

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

[31] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[32] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[33] S. Vongehr and X. Meng, "The missing memristor has not been found," *Scientific reports*, vol. 5, p. 11657, 2015.

[34] L. Torres, R. M. Brum, L. V. Cargnini, and G. Sassatelli, "Trends on the application of emerging nonvolatile memory to processors and programmable devices," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pp. 101–104, IEEE, 2013.

[35] D. L. Lewis and H.-H. S. Lee, "Architectural evaluation of 3d stacked rram caches," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pp. 1–4, IEEE, 2009.

[36] X. Zhu, X. Yang, C. Wu, N. Xiao, J. Wu, and X. Yi, "Performing stateful logic on memristor memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 10, pp. 682–686, 2013.

[37] M. Laiho and E. Lehtonen, "Arithmetic operations within memristor-based analog memory," in *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pp. 1–4, IEEE, 2010.

[38] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," *Physical Review E*, vol. 84, no. 4, p. 046703, 2011.

[39] E. Gale, B. de Lacy Costello, and A. Adamatzky, "Boolean logic gates from a single memristor via low-level sequential logic," in *International Conference on Unconventional Computing and Natural Computation*, pp. 79–89, Springer, 2013.

[40] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*, pp. 759–766, IEEE, 2016.

[41] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1718–1725, EDA Consortium, 2015.

[42] J. De Kleer and G. J. Sussman, "Propagation of constraints applied to circuit synthesis," *International Journal of Circuit Theory and Applications*, vol. 8, no. 2, pp. 127–144, 1980.

[43] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "A mixed-integer nonlinear programming approach to analog circuit synthesis," in *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*, pp. 698–703, IEEE, 1992.

[44] K. Antreich, J. Eckmueller, H. Graeb, M. Pronath, F. Schenkel, R. Schwencker, and S. Zizala, "Wicked: Analog circuit synthesis incorporating mismatch," in *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pp. 511–514, IEEE, 2000.

[45] J. D. Lohn and S. P. Colombano, "Automated analog circuit synthesis using a linear representation," in *International Conference on Evolvable Systems*, pp. 125–133, Springer, 1998.

[46] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *IEEE journal of solid-state circuits*, vol. 31, no. 6, pp. 792–803, 1996.

[47] M. Ranjan, W. Verhaegen, A. Agarwal, H. Sampath, R. Vemuri, and G. Gielen, "Fast, layout-inclusive analog circuit synthesis using pre-compiled parasitic-aware symbolic performance models," in *Proceedings of the conference on Design, automation and test in Europe-Volume 1*, p. 10604, IEEE Computer Society, 2004.

[48] N. K. Jha and S.-J. Wang, "Design and synthesis of self-checking vlsi circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, pp. 878–887, 1993.

[49] H. Wang and S. B. Vrudhula, "Behavioral synthesis of field programmable analog array circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 4, pp. 563–604, 2002.

[50] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 710–722, 2003.

[51] M. Saeedi, M. S. Zamani, M. Sedighi, and Z. Sasanian, "Reversible circuit synthesis using a cycle-based approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 6, no. 4, p. 13, 2010.

[52] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, 2006.

[53] W. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1652–1663, 2006.

[54] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 796–801, ACM, 2012.

[55] S. Ghosh, S. Bhunia, and K. Roy, "Crista: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 1947–1956, 2007.

[56] T. Reis, "Circuit synthesis of passive descriptor systems—a modified nodal approach," *International Journal of Circuit Theory and Applications*, vol. 38, no. 1, pp. 44–68, 2010.

[57] J. R. Koza, D. Andre, F. H. Bennett III, and M. A. Keane, "Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming," in *Proceedings of the 1st annual conference on genetic programming*, pp. 132–140, MIT Press, 1996.

[58] M. Kubica, A. Opara, and D. Kania, "Logic synthesis for fpgas based on cutting of bdd," *Microprocessors and Microsystems*, 2017.

[59] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: Pushing power management to logic synthesis/design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 1051–1060, 1998.

[60] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 72, no. 5, pp. 593–599, 1953.

[61] D. Zeheb and W. Caywood, "A symbolic method for synthesis of 2-terminal switching circuits," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 73, no. 6, pp. 690–693, 1955.

[62] E. J. McCluskey, *Algebraic minimization and the design of two-terminal contact networks*. PhD thesis, Massachusetts Institute of Technology, 1956.

[63] D. Whitehead, "Algorithm for logic-circuit synthesis by using multiplexers," *Electronics Letters*, vol. 13, no. 12, pp. 355–356, 1977.

[64] T. Sasao, "Macdas: Multi-level and-or circuit synthesis using two-variable function generators," in *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pp. 86–93, IEEE Press, 1986.

[65] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sis: A system for sequential circuit synthesis," 1992.

[66] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Computer Design: VLSI in Computers and Processors, 1992. ICCD'92. Proceedings, IEEE 1992 International Conference on*, pp. 328–333, IEEE, 1992.

[67] L. Jóźwiak, "General decomposition and its use in digital circuit synthesis," *VLSI Design*, vol. 3, no. 3-4, pp. 225–248, 1995.

[68] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[69] J. Gu and R. Puri, "Asynchronous circuit synthesis with boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 961–973, 1995.

[70] M. Renaudin, "Asynchronous circuits and systems: a promising design alternative," *Microelectronic engineering*, vol. 54, no. 1-2, pp. 133–149, 2000.

[71] A. Bardsley and D. Edwards, *Balsa: An asynchronous circuit synthesis system*. University of Manchester, 1998.

[72] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 63–73, 1991.

[73] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 783–789, 1997.

[74] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli, "Logic synthesis for large pass transistor circuits," in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pp. 663–670, IEEE Computer Society, 1997.

[75] K. Roy and S. Prasad, "Syclop: Synthesis of cmos logic for low power applications," in *Computer Design: VLSI in Computers and Processors, 1992. ICCD'92. Proceedings, IEEE 1992 International Conference on*, pp. 464–467, IEEE, 1992.

[76] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: A fresh look at combinational logic synthesis," in *Design Automation Conference, 2006 43rd ACM/IEEE*, pp. 532–535, IEEE, 2006.

[77] A. Sawa, "Resistive switching in transition metal oxides," *Materials today*, vol. 11, no. 6, pp. 28–36, 2008.

[78] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[79] J. P. Strachan, J. J. Yang, R. Münstermann, A. Scholl, G. Medeiros-Ribeiro, D. R. Stewart, and R. S. Williams, "Structural and chemical characterization of tio2 memristive devices by spatially-resolved nexafs," *Nanotechnology*, vol. 20, no. 48, p. 485701, 2009.

[80] M. Fujimoto, H. Koyama, M. Konagai, Y. Hosoi, K. Ishihara, S. Ohnishi, and N. Awaya, "TiO_2 anatase nanolayer on TiN thin film exhibiting high–speed bipolar resistive switching," *Applied Physics Letters*, vol. 89, no. 22, p. 223509, 2006.

[81] C. Schindler, S. C. P. Thermadam, R. Waser, and M. N. Kozicki, "Bipolar and unipolar resistive switching in Cu-doped SiO_2," *IEEE Transactions on Electron Devices*, vol. 54, no. 10, pp. 2762–2768, 2007.

[82] C.-Y. Lin, C.-Y. Wu, C.-Y. Wu, T.-C. Lee, F.-L. Yang, C. Hu, and T.-Y. Tseng, "Effect of top electrode material on resistive switching properties of ZrO_2 film memory devices," *IEEE Electron Device Letters*, vol. 28, no. 5, pp. 366–368, 2007.

[83] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nature materials*, vol. 6, no. 11, pp. 833–840, 2007.

[84] Q. Liu, W. Guan, S. Long, R. Jia, M. Liu, and J. Chen, "Resistive switching memory effect of zr o 2 films with zr+ implanted," *Applied physics letters*, vol. 92, no. 1, p. 012117, 2008.

[85] Y. Dong, G. Yu, M. C. McAlpine, W. Lu, and C. M. Lieber, "Si/a-si core/shell nanowires as nonvolatile crossbar switches," *Nano Letters*, vol. 8, no. 2, pp. 386–391, 2008.

[86] W. Guan, S. Long, Q. Liu, M. Liu, and W. Wang, "Nonpolar nonvolatile resistive switching in cu doped zro_2," *IEEE Electron Device Letters*, vol. 29, no. 5, pp. 434–437, 2008.

[87] U. Russo, D. Ielmini, C. Cagli, and A. L. Lacaita, "Filament conduction and reset mechanism in nio-based resistive-switching memory (rram) devices," *IEEE Transactions on Electron Devices*, vol. 56, no. 2, pp. 186–192, 2009.

[88] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, p. 074508, 2009.

[89] J. Borghetti, D. B. Strukov, M. D. Pickett, J. J. Yang, D. R. Stewart, and R. S. Williams, "Electrical transport and thermometry of electroformed titanium dioxide memristive switches," *Journal of Applied Physics*, vol. 106, no. 12, p. 124504, 2009.

[90] S. Lee, H. Kim, D.-J. Yun, S.-W. Rhee, and K. Yong, "Resistive switching characteristics of zno thin film grown on stainless steel for flexible nonvolatile memory devices," *Applied physics letters*, vol. 95, no. 26, p. 262113, 2009.

[91] J. J. Yang, M.-X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams, "High switching endurance in tao x memristive devices," *Applied Physics Letters*, vol. 97, no. 23, p. 232102, 2010.

[92] S. H. Jo, K.-H. Kim, T. Chang, S. Gaba, and W. Lu, "Si memristive devices applied to memory and neuromorphic circuits," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 13–16, IEEE, 2010.

[93] S. K. Hong, J. E. Kim, S. O. Kim, S.-Y. Choi, and B. J. Cho, "Flexible resistive switching memory device based on graphene oxide," *IEEE Electron device letters*, vol. 31, no. 9, pp. 1005–1007, 2010.

[94] R. Muenstermann, T. Menke, R. Dittmann, and R. Waser, "Coexistence of filamentary and homogeneous resistive switching in fe-doped srtio3 thin-film memristive devices," *Advanced materials*, vol. 22, no. 43, pp. 4819–4822, 2010.

[95] D.-H. Kwon, K. M. Kim, J. H. Jang, J. M. Jeon, M. H. Lee, G. H. Kim, X.-S. Li, G.-S. Park, B. Lee, S. Han, *et al.*, "Atomic structure of conducting nanofilaments in tio2 resistive switching memory," *Nature nanotechnology*, vol. 5, no. 2, pp. 148–153, 2010.

[96] K. H. Choi, M. Mustafa, K. Rahman, B. K. Jeong, and Y. H. Doh, "Cost-effective fabrication of memristive devices with zno thin film using printed electronics technologies," *Applied Physics A*, vol. 106, no. 1, pp. 165–170, 2012.

[97] Y. Wu, S. Yu, B. Lee, and P. Wong, "Low-power tin/al2o3/pt resistive switching device with sub-20 μa switching current and gradual resistance modulation," *Journal of Applied Physics*, vol. 110, no. 9, p. 094104, 2011.

[98] K.-L. Lin, T.-H. Hou, J. Shieh, J.-H. Lin, C.-T. Chou, and Y.-J. Lee, "Electrode dependence of filament formation in hfo2 resistive-switching memory," *Journal of Applied Physics*, vol. 109, no. 8, p. 084104, 2011.

[99] L. Gao, F. Alibart, and D. B. Strukov, "Analog-input analog-weight dot-product operation with ag/a-si/pt memristive devices," in *VLSI and System-on-Chip, 2012 (VLSI-SoC), IEEE/IFIP 20th International Conference on*, pp. 88–93, IEEE, 2012.

[100] Y. Yang, P. Sheridan, and W. Lu, "Complementary resistive switching in tantalum oxide-based resistive memory devices," *Applied Physics Letters*, vol. 100, no. 20, p. 203112, 2012.

[101] A. Mehonic, S. Cueff, M. Wojdak, S. Hudziak, O. Jambois, C. Labbé, B. Garrido, R. Rizk, and A. J. Kenyon, "Resistive switching in silicon suboxide films," *Journal of Applied Physics*, vol. 111, no. 7, p. 074507, 2012.

[102] S. Pi, P. Lin, and Q. Xia, "Cross point arrays of 8 nm$\times$ 8 nm memristive devices fabricated with nanoimprint lithography," *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 31, no. 6, p. 06FA02, 2013.

[103] R. Sakdanuphab and A. Sakulkalavek, "Resistive switching behavior of ti/zno/mo thin film structure for nonvolatile memory applications," in *Key Engineering Materials*, vol. 659, pp. 588–592, Trans Tech Publ, 2015.

[104] D. Kuzmichev and Y. Y. Lebedinskii, "Resistive switching in mim structure based on over-stoichiometric tantalum oxide," *Microelectronic Engineering*, vol. 178, pp. 150–153, 2017.

[105] C. Sun, S. Lu, F. Jin, W. Mo, J. Song, and K. Dong, "Control the switching mode of pt/hfo2/tin rram devices by tuning the crystalline state of tin electrode," *Journal of Alloys and Compounds*, vol. 749, pp. 481–486, 2018.

[106] R. Prakash, S. Sharma, A. Kumar, and D. Kaur, "Improved resistive switching performance in cu-cation migrated mos2 based reram device incorporated with tungsten nitride bottom electrode," *Current Applied Physics*, vol. 19, no. 3, pp. 260–265, 2019.

[107] W. H. Ng, A. Mehonic, M. Buckwell, L. Montesi, and A. J. Kenyon, "High-performance resistance switching memory devices using spin-on silicon oxide," *IEEE Transactions on Nanotechnology*, vol. 17, no. 5, pp. 884–888, 2018.

[108] Z.-Y. He, T.-Y. Wang, L. Chen, H. Zhu, Q.-Q. Sun, S.-J. Ding, and D. W. Zhang, "Atomic layer-deposited hfalox-based rram with low operating voltage for computing in-memory applications," *Nanoscale research letters*, vol. 14, no. 1, p. 51, 2019.

[109] K. M. Kim, D. S. Jeong, and C. S. Hwang, "Nanofilamentary resistive switching in binary oxide system; a review on the present statusand outlook," *Nanotechnology*, vol. 22, no. 25, p. 254002, 2011.

[110] J. Qiu, A. Shih, W. Zhou, Z. Mi, and I. Shih, "Effects of metal contacts and dopants on the performance of zno-based memristive devices," *Journal of Applied Physics*, vol. 110, no. 1, p. 014513, 2011.

[111] D. Strukov and H. Kohlstedt, "Resistive switching phenomena in thin films: Materials, devices, and applications," *MRS bulletin*, vol. 37, no. 2, pp. 108–114, 2012.

[112] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.

[113] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–5878, 2013.

[114] A. Siemon, T. Breuer, N. Aslam, S. Ferch, W. Kim, J. van den Hurk, V. Rana, S. Hoffmann-Eifert, R. Waser, S. Menzel, *et al.*, "Realization of boolean logic functionality using redox-based memristive devices," *Advanced functional materials*, vol. 25, no. 40, pp. 6414–6423, 2015.

[115] D. Kumar, R. Aluguri, U. Chand, and T. Tseng, "Metal oxide resistive switchung memory: Materials, properties and switching mechanisms," *Ceramics International*, 2017.

[116] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016. US Patent 9,319,047.

[117] A. Velasquez, *Computation of Boolean Formulas Using Sneak Paths in Crossbar Computing*. PhD thesis, University of Central Florida, 2014.

[118] A. Velasquez and S. K. Jha, "Automated synthesis of crossbars for nanoscale computing using formal methods," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pp. 130–136, IEEE, 2015.

[119] 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 *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pp. 1870–1873, IEEE, 2016.

[120] A. Velasquez and S. K. Jha, "Parallel computing using memristive crossbar networks: Nullifying the processor-memory bottleneck," in *Design & Test Symposium (IDT), 2014 9th International*, pp. 147–152, IEEE, 2014.

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

[122] P. J. Kuekes, R. S. Williams, and J. R. Heath, "Molecular wire crossbar memory," Oct. 3 2000. US Patent 6,128,214.

[123] Y. Chen and R. S. Williams, "Configurable nanoscale crossbar electronic circuits made by electrochemical reaction," Feb. 11 2003. US Patent 6,518,156.

[124] N. A. Melosh, A. Boukai, F. Diana, B. Gerardot, A. Badolato, P. M. Petroff, and J. R. Heath, "Ultrahigh-density nanowire lattices and circuits," *Science*, vol. 300, no. 5616, pp. 112–115, 2003.

[125] G.-Y. Jung, E. Johnston-Halperin, W. Wu, Z. Yu, S.-Y. Wang, W. M. Tong, Z. Li, J. E. Green, B. A. Sheriff, A. Boukai, *et al.*, "Circuit fabrication at 17 nm half-pitch by nanoimprint lithography," *Nano Letters*, vol. 6, no. 3, pp. 351–354, 2006.

[126] S. Pi, C. Li, H. Jiang, W. Xia, H. Xin, J. J. Yang, and Q. Xia, "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension," *Nature nanotechnology*, vol. 14, no. 1, p. 35, 2019.

[127] M. A. Rothman, V. J. Zimmer, G. P. Mudusuru, J. Yao, and J. Lin, "Technology to facilitate rapid booting with high-speed and low-speed nonvolatile memory," Oct. 11 2018. US Patent App. 15/484,513.

[128] M. Rothman and V. Zimmer, "Low latency boot from zero-power state," Feb. 7 2019. US Patent App. 15/891,124.

[129] C.-Y. Lee, "Representation of switching circuits by binary-decision programs," *Bell Labs Technical Journal*, vol. 38, no. 4, pp. 985–999, 1959.

[130] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[131] F. Somenzi, "Cudd: Cu decision diagram package release 3.0. 0," *University of Colorado at Boulder*, 2015.

[132] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 770–775, IEEE, 2017.

[133] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15, Springer, 1987.

[134] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*, vol. 185. IOS press, 2009.

[135] D. Chakraborty and S. K. Jha, "Design of compact memristive in-memory computing systems using model counting," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pp. 1–4, IEEE, 2017.

[136] D. Chakraborty, S. Raj, and S. K. Jha, "A compact 8-bit adder design using in-memory memristive computing: Towards solving the feynman grand prize challenge," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 67–72, IEEE, 2017.

[137] K.-S. Li, C. Ho, M.-T. Lee, M.-C. Chen, C.-L. Hsu, J. Lu, C. Lin, C. Chen, B. Wu, Y. Hou, *et al.*, "Utilizing sub-5 nm sidewall electrode technology for atomic-scale resistive memory fabrication," in *VLSI Technology (VLSI-Technology): Digest of Technical Papers, 2014 Symposium on*, pp. 1–2, IEEE, 2014.

[138] D. Chakraborty, S. Raj, J. C. Gutierrez, T. Thomas, and S. K. Jha, "In-memory execution of compute kernels using flow-based memristive crossbar computing," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[139] C. Lattner and V. Adve, "The llvm compiler framework and infrastructure tutorial," in *International Workshop on Languages and Compilers for Parallel Computing*, pp. 15–16, Springer, 2004.

[140] S. Chakraborti, P. V. Chowdhary, K. Datta, and I. Sengupta, "Bdd based synthesis of boolean functions using memristors," in *Design & Test Symposium (IDT), 2014 9th International*, pp. 136–141, IEEE, 2014.

[141] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pp. 948–953, EDA Consortium, 2016.

[142] B. J. Choi, A. C. Torrezan, J. P. Strachan, P. Kotula, A. Lohn, M. J. Marinella, Z. Li, R. S. Williams, and J. J. Yang, "High-speed and low-energy nitride memristors," *Advanced Functional Materials*, vol. 26, no. 29, pp. 5290–5296, 2016.

[143] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

[144] P. Nenzi and H. Vogt, "Ngspice 26," 2013.

[145] S. Hutchinson, E. Keiter, R. Hoekstra, H. Watts, A. Waters, T. Russo, R. Schells, S. WIX, and C. BOGDAN, "The xyce^{TM} parallel electronic simulator–an overview," in *Parallel Computing: Advances and Current Issues*, pp. 165–172, World Scientific, 2002.