Memristive Device for Logic Design and Computing

Xie, Lei

DOI
10.4233/uuid:52c58e54-883a-4268-8413-c7491dc78671

Publication date
2018

Document Version
Final published version

Citation (APA)
Xie, L. (2018). *Memristive Device for Logic Design and Computing*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:52c58e54-883a-4268-8413-c7491dc78671

MEMRISTIVE DEVICE FOR LOGIC DESIGN AND COMPUTING

MEMRISTIVE DEVICE FOR LOGIC DESIGN AND COMPUTING

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 26 februari 2018 om 15:00 uur

door

Lei XIE

Master of Engineering in Electronic Science & Technology,
Xi'an Jiaotong University, Xi'an, China,
geboren te Yinchuan, Ningxia, China.

This dissertation has been approved by the

promotor: Prof. dr. ir. S. Hamdioui

Composition of the doctoral committee:

Rector Magnificus,	chairman
Prof. dr. ir. S. Hamdioui,	Delft University of Technology, promotor
Independent members:	
Prof. dr. ir. K.L.M. Bertels,	Delft University of Technology
Dr. ir. T.G.R.M. van Leuken,	Delft University of Technology
Prof. dr. phil.nat.habil. R. Tetzlaff,	TU Dresden, Germany
Prof. dr. ir. A.J. van der Veen,	Delft University of Technology
Prof. dr. H. Corporaal,	Eindhoven University of Technology
Dr. ir. H.G. Kerkhoff,	University of Twente

Keywords: Memristor, Logic, Computing

ISBN 978-94-6366-013-6
This research was financially supported by China Scholarship Council (CSC)

CONTENTS

SUMMARY

Memristive device or memristor is a promising emerging technology due to its good scalability, near-zero standby power consumption, high integration density, and CMOS fabrication compatibility. Several potential applications based on memristor technology have been proposed, such as non-volatile memories, neuromorphic systems, and resistive computing. However, research on resistive computing is still in its infancy phase. Therefore, it faces challenges with respect to the development of the device technology, logic design styles, computer architectures, compilers and applications.

This thesis focuses on the logic design (including primitive logic gates, interconnect, circuit, and synthesis flow) and a novel non-Von Neumann architecture.

Primitive logic gate – We first explore the complete logic gate space for Snider logic. Subsequently, we develop a novel logic design style referred to as scouting logic; it performs logic operations by modifying standard memory read operations. In addition, we analyze robustness of logic gates while considering memristive device variability, parasitic resistors and capacitors of nanowires, sneak path currents, and different memristor models. Despite the contribution of this thesis, innovative design styles still need to be explored and more research is required in designing robust logic circuits against device variability.

Interconnect design – We explore and compare three approaches to implement general interconnect schemes; they are using only memristor crossbar, CMOS peripheral circuits and a hybrid of memristor crossbar and pass transistors. Next, we explore the intra-tile and inter-tile communication schemes using memristor crossbar. In addition, we further explore the possibility to use dedicated interconnect schemes to address specific algebraic problems, such as matrix transpose. It worth to note that more efforts are required to generalize and optimize the communication infrastructure automatically.

Circuit design and synthesis flow – We develop methodologies to design ASICs and FPGAs using memristor logic design styles. For ASICs, we first explore the place-and-route methods for large-scale circuits, and then develop a synthesis flow and the related evaluation model. For FPGAs, we develop two different implementations based on memristor logic, and we automate their design and evaluation flow. We observe that both ASICs and FPGAs based on memristor logic suffer from the CMOS control parts. An intelligent CMOS controller is therefore essential for overall improvements.

Non-Von Neumann architecture – We explore a non-von Neumann architecture, which is referred to as Computation-In-Memory (CIM Architecture), for specific data-intensive applications. CIM architecture integrates both storage and processing elements in the

same physical location using memristor technology; hence, it significantly reduces the memory access time and energy consumption. The preliminary results show that CIM architecture obtains significant improvements (e.g., energy-delay product and computational efficiency) over conventional multi-core architectures for specific applications (e.g., parallel addition, DNA sequencing).

Het geheugenresistieve element of de geheugenweerstand is een veelbelovende opkomende technologie vanwege de goede schaalbaarheid, bijna-nul stand-by stroomverbruik, hoge integratiedichtheid en compatibiliteit met het CMOS-fabricage proces. Er zijn al verschillende mogelijke toepassingen op basis van geheugenresistieve technologie voorgesteld, waaronder niet-vluchtige geheugens, neuromorfe systemen en resistieve gegensverwerking. Onderzoek naar geheugenresistieve gegensverwerking staat echter nog in de kinderschoenen. Daarom staat het voor uitdagingen met betrekking tot de ontwikkeling van de elementtechnologie, logische ontwerpstijlen, computerarchitecturen, compilers en toepassingen.

Dit proefschrift richt zich op het logische ontwerp (waaronder primitieve logische poorten, onderlinge verbindingen, schakelingen en synthese) en een nieuwe niet-Von Neumann-architectuur.

Primitieve logische poort – We onderzoeken eerst de complete logische poortruimte voor Snider-logica. Vervolgens ontwikkelen we een nieuwe logische-ontwerpstijl genaamd scouting-logica; het voert logische operaties uit door standard geheugen-leesoperaties aan te passen. Bovendien, analyseren we de robuustheid van de logische poorten rekening houdende met de elementvariabiliteit, parasitaire weerstanden en capaciteiten van de nanodraden, sluipbaanstromen en verschillende geheugenweerstandmodellen. Ondanks de bijdrage van dit proefschrift dienen innovatieve ontwerpstijlen nog verder onderzocht te worden en is meer onderzoek vereist voor het ontwerpen van robuuste logische schakelingen tegen elementvariabiliteit.

Onderlinge-verbindingsontwerp – We onderzoeken en vergelijken drie methodes om algemene onderlinge-verbindingsschema's te implementeren; ze gebruiken enkel de geheugenweerstandskruisschakeling, CMOS-hulpschakelingen en een hybride van de geheugenweerstandskruisschakeling en doorgeeftransistors. Vervolgens, onderzoeken we de intra-tegel en inter-tegel communicatiemethodes voor de geheugenweerstandskruisschakeling. Bovendien onderzoeken we de mogelijkheid om toegewijde onderlinge-verbindingsschema's te gebruiken om specifieke algebraïsche problemen te adresseren, zoals matrixtranspositie. Het is de moeite waard te vermelden dat meer inspanning vereist is om de communicatie-infrastructuur automatisch te generaliseren en optimaliseren.

Schakelingontwerp en synthese stappenplan – We ontwikkelen methodologieën voor het ontwerpen van ASIC's en FPGA's die gebruikmaken van geheugenresistieve logische-ontwerpstijlen. Voor ASIC's onderzoeken we eerst plaats-en-verbind-methodes voor grootschalige schakelingen en vervolgens ontwikkelen we een synthese stappenplan en het bijbehorende evaluatiemodel. Voor FPGA's ontwikkelen we twee verschillende implementaties gebaseerd op geheugenresistieve logica en automatiseren we de ontwerp- en evaluatiestappenplannen. We merken op dat beide op ASIC- en FPGA-gebaseerde geheugenresistieve logica lijden onder de CMOS-aansturingsonderdelen. Een intelligente CMOS-regelaar is daarom essentieel voor een algehele verbetering.

Niet-Von Neumann-architectuur – We onderzoeken een niet-Von Neumann-architectuur, genaamd Gegevensverwerking-in-Geheugen (GiG-architectuur) voor specifieke data-intensieve toepassingen. De GiG-architectuur integreert zowel opslag- en verwerkingselementen in dezelfde fysieke locatie met behulp van geheugenresistieve technlogie; daarom vermindert het de geheugentoegangsstijd en het energieverbruik. De voorlopige resultaten tonen dat de GiG-architectuur significante verbeteringen brengt (bijvoorbeeld in het energie-vertraging product en de verwerkingsefficiëntie) ten opzichte van conventionele multikernarchitecturen voor specifieke toepassingen (bijvoorbeeld parallelle optellingen en DNA-sequentiereactie).

ACKNOWLEDGEMENTS

After four years, I can finally say that my Ph.D. dissertation has ended. It has been a unique experience with many ups and downs. Fortunately, I have got through everything with the help from my professors, colleagues, friends and families.

First of all, I would like to thank my promotor Prof. dr. ir. S.Hamdioui, for providing me the opportunity to pursue my Ph.D. thesis under his guidance. Not did I only learn how to do research, write scientific papers, present my work, but also what it means to be a team worker and collaborate with my colleagues. Thanks for your continuous motivation and the appropriate guidance during my Ph.D. study. In addition to properly educating me as an independent researcher, you gave me many valuable chances to develop my skills more than research such as being a teaching assistant, organizing international conferences, etc. Thank you very much for all your efforts, patience and helps during my Ph.D. study. I also would like to thank Prof. dr. ir. K.L.M. Bertels. As the head of our Quantum and Computer Engineering Department, you offers us a wonderful and friendly atmosphere. You have been encouraging us to attend social events, such as football, bowling, etc. I would also like to thank the remaining committee members for accepting their role, reading this dissertation, and providing feedback; thank you for all your efforts. In addition, I would like to thank Motta and Daniel for translating my thesis summary and propositions into Dutch.

I would like to thank the CE secretaries and staff for taking care of all the management and technical supports related to my day-to-day work. Lidwina and Joyce, thank you for managing all the paperworks and other secretary-related tasks. Erik, thank you for creating and keeping the websites updated, managing the servers, fixing computer problems, installing various software, etc.

Hoang Anh, Jintao, dr. Mottaqiallah Taouil and Prof. dr. ir. S. Hamdioui, thank you for your contributions as members of the CE memristor team. The brainstorm sessions have been very helpful. You have built a great team to work with, always helpful, and open to discussions. Specially, I would like to appreciate dr. Taouil for his helpful discussions and paper revisions. Also, I would like to thank our previous memristor team members, Adib and Berna.

I would like to thank Imran for organizing the CE weekly football games. These games provided me good chances to refresh myself and recharge my energy to continue the challenging tasks during my Ph.D. Everyone that participated in these matches, thank you all! Also, special thanks to George, Mihai, Hoang Anh, Joost and Leon for organizing various other CE social events. Everyone that participated and contributed to the nice atmosphere, thank you as well.

I would like to extend my thanks to my previous and current office mates. Anthony, Hoang Anh, Innocent, Adib, Berna and Jintao. Thank you for the many discussions and interesting chats. I thank each of you individually for the pleasant working environment.

I would like to thank Shanshan, Xiang, Jian, Jintao, Lingling, Yande, and Lizhou for our close friendship on the campus and in particular for the enjoyable daily lunches we had. I really also appreciate your delicious dinners and interesting board games after them. I would like to also thank all the other Chines friends, Shi, Qi, Sensen and his family, Long, Xiaohui, Jingtang, Lian, Tian, Yan. I would also like to thanks Anthony, Hoang Anh, Joost and Jintao for going restaurant for enjoying some delicious food.

Last but not least, I would like to express my deepest thanks to my family for all the support they gave me, in particular my mother and father! My mother and father, there are no words that can fully express my gratitude towards you. Love you always.

Lei Xie

Delft, February, 2018
the Netherlands

1

INTRODUCTION

Downscaling of CMOS technology has been approaching the device physical limits. Therefore, the conventional CMOS technology is suffering from the magnificent challenges, e.g., increased static power consumption, saturated clock frequency, reduced device reliability, and a more complex manufacturing process. In order to address such challenges, novel device technologies (e.g., memristive device, carbon nanotube, etc.) have been under research as alternatives to the future very large scale integration circuits. Among them, memristive device (or memristor) is a promising candidate due to its great scalability, high integration density, near-zero standby power consumption, and CMOS fabrication compatibility. Many potential applications of memristive device technology have been proposed, such as non-volatile memories, neuromorphic processing, logic gates and resistive computing paradigms.

This chapter first introduces the motivation behind resistive computing paradigms. Thereafter, it presents the opportunities and challenges of resistive computing. Subsequently, it briefly describes the research topics of this thesis followed by its main contributions. Finally, it provides the organization of the remainder of this thesis.

1

1.1. MOTIVATION

Emerging applications, such as big data and artificial intelligence, require exascale computing capabilities (i.e., 10^{18} calculations per second) [1–3]. Such applications not only significantly influence our daily life, but also change the computer science and semiconductor industry deeply.

Unfortunately, both today's computer architectures and device technologies are encountering major challenges, which make them incapable of delivering the required computing power. On one hand, the performance of today's computer architectures are limited by the three well-known walls [1,2,4], as shown in Fig. 1.1 and explained next.

- **Memory wall** is caused by the increasing speed gap between processor and memory. Consequently, the limited memory bandwidth makes memory accesses the performance killer, as shown by the saturated single-thread performance of Fig. 1.1.
- **Power wall** is reached due to a power limit for cooling as shown in Fig. 1.1. As a result, the CPU clock frequency cannot increase further, and hence the performance of a single-core CPU stagnated (see Fig. 1.1).
- **Instruction Level parallelism (ILP) wall** is reached due to the increasing difficulty of extracting enough parallelisms for multi-core applications. Consequently, processing resources remain idle and hence it makes the increasing number of cores further not attractive (see Fig. 1.1).

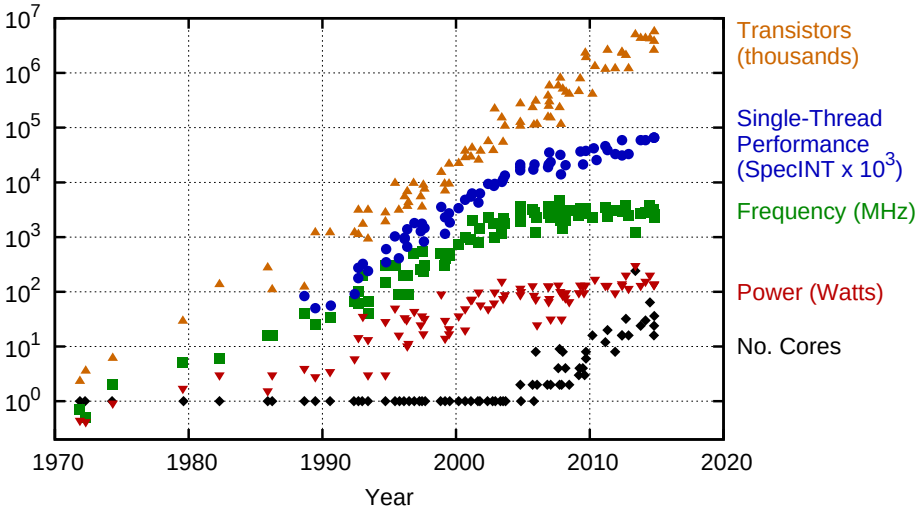


Figure 1.1: Trends of Microprocessors in Last 40 Years [1,4]

On the other hand, CMOS technology, which has been the main driving force of today's computer industry, is also facing three walls [2,4,5].

- **Reliability wall** is occurring as the down scaling of CMOS technology is approaching the physical device limits. Consequently, a CMOS transistor has a shorter lifetime and higher failure rate.
- **Leakage wall** is becoming more apparent with technology scaling due to the volatile CMOS technology and lower threshold voltages used for CMOS. As a consequence, the static power is increasing as a dominant part.
- **Cost wall** is caused by adding new materials, masks and process steps to newer technology nodes. As a consequence, this reduces the cost benefits purely obtained from geometric scaling.

All the above walls have slowed down advancements of both traditional computer architectures and CMOS technology. To address these walls, alternative computing paradigms based on novel device technologies must be explored [5–10]. Resistive computing, neuromorphic computing and quantum computing are several candidates for the next-generation computing paradigms, while memristor, quantum dots, spin-wave devices are a couple of emerging device technologies. Resistive computing is promising candidate as it is able to process data within memory, and provide high energy efficiency and massive parallelism [11–16]. Memristive device is a promising candidate to (partially) replace traditional CMOS technology (at least in some applications) due to many advantages including great scalability and high density, CMOS process compatibility, near-zero standby power, and its potential to implement new computing paradigms [5,17].

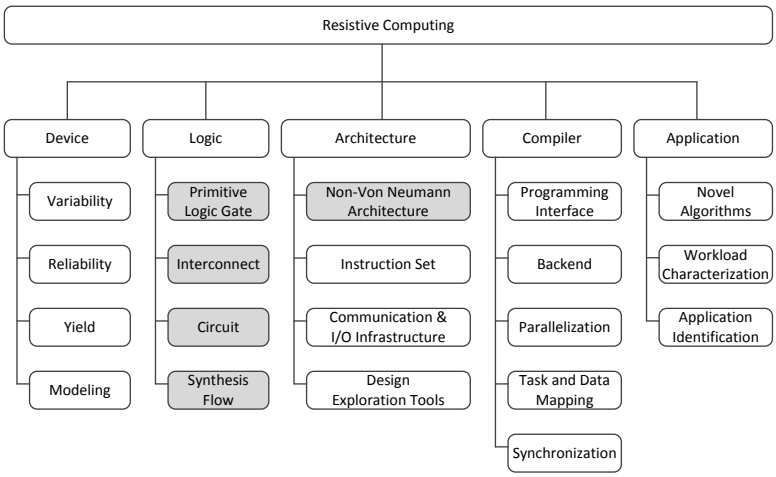


Figure 1.2: Overview of Resistive Computing.

1.2. OPPORTUNITIES AND CHALLENGES

This section briefly overviews both the opportunities and challenges of resistive computing. Fig. 1.2 shows the different aspects that have to be explored in order to implement

resistive computing architectures; these consist of device, logic, architecture, compiler and application. Each aspect is further described below.

Device: Resistive computing architectures use resistive switching devices for computation, such as Resistive RAM (RRAM), Conductive Bridge RAM (CBRAM), Phase Change Memory (PCM), Spin-Transfer Torque Magnetic RAM (STT-MRAM) [18]. These devices are also referred to as memristive devices or memristors [19,20]. Memristor technology is a promising alternative for CMOS technology due to its many advantages including CMOS process compatibility, near-zero standby power, great scalability, high integration density and enabling both logic and memories for computation [5,17]. Memristor technology has been studied by both academia and industry (e.g., Samsung, Micron, Toshiba and Sony) as shown in Fig. 1.3; the capacity of these memory prototypes are growing continuously in recent years up to a volume of 32 Gb. However, memristor technology is also facing major challenges in terms of reliability (e.g., limited endurance), variability (e.g., cycle-to-cycle variation), low yield and appropriate device modeling [17,19,21] (see Fig. 1.2).

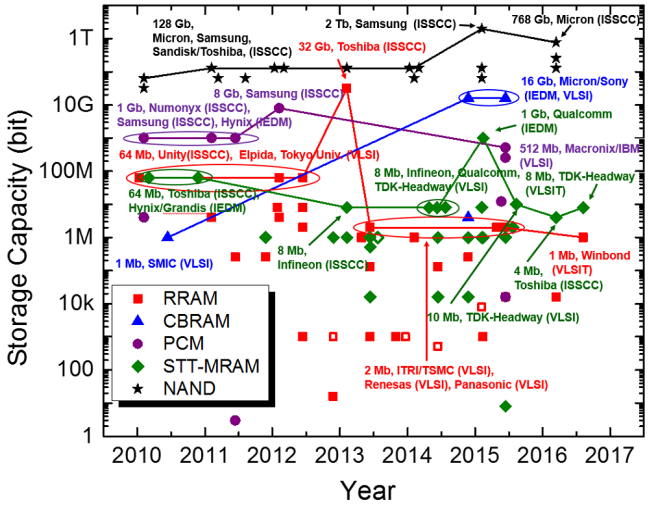


Figure 1.3: Trends of Emerging Device Technologies [18]

Logic: Many memristor based logic design styles [15,22–36] have been proposed recently. They can be classified using the following criteria:

- **Input Data Representation** indicates whether the input data are represented by a resistance or voltage.
- **Output Data Representation** indicates whether the output data are represented by a resistance or voltage.

- **Processing Element** indicates whether the data is processed by memristors only or a hybrid of CMOS transistors and memristors.

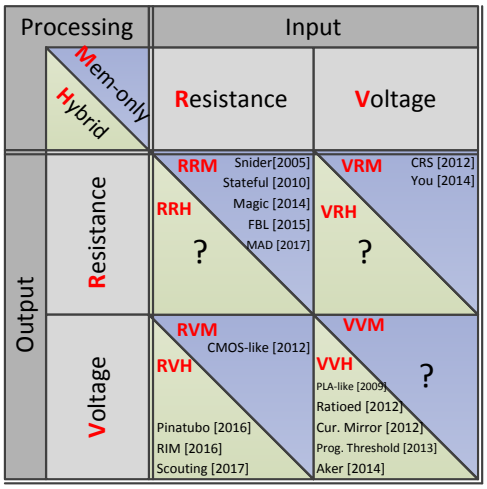


Figure 1.4: Overview of Logic Design Styles [37]

Fig. 1.4 shows the classification result. Each class is named using the input and output representation signals and the processing element. For instance, scouting logic is part of the RVH class where R represents input resistance, V output voltage and H hybrid CMOS/memristor processing. The existing logic design styles are parts of the five out of eight classes. In *RRM*, logic design styles use resistances to represent both the input and output data. They perform logic operations using memristors to form voltage dividers which conditionally switches the output memristors. In *VRM*, logic design styles use a voltage to represent the input data, while a resistance to represent the output data. They perform logic operations by modifying the standard memory write operations. In *RVM*, the logic design style uses a resistance to represent the input data, while a voltage to represent the output data. It performs logic operations by replacing the transistors in the pull-up and pull-down networks of the traditional CMOS logic with memristors where they function as digital switches. In *RVH*, logic design styles use a resistance to represent the input data while a voltage to represent the output data. They perform logic operations by modifying the standard memory read operations. In *VVH*, logic design styles use voltages to represent both the input and output data, while use CMOS gates (e.g., inverter or D Flip-Flop) as a threshold function. The memristors are used as either configuration switches or input weights.

Memristor based logic circuits have an immense potential to reduce area and power consumption, as memristor technology provides a great scalability (5 nm), a high integration density (10GB/cm^2) and near-zero standby power [17,19,21]. In addition, its unique characteristics (i.e., multi-level state and non-volatility) also inspire and motivate novel circuits (e.g., dot matrix multiplication [38], high-radix arithmetic circuits [39,40]). How-

ever, memristor logic circuits also face major challenges [37], including development of innovative primitive logic gates, appropriate interconnect schemes, efficient arithmetic circuits and automated synthesis flows. In addition, it is crucial to properly evaluate the impact of device variability and sneak path currents on the circuit robustness [37,41,42].

Architecture: The ability to use memristors both for memories and logic has led to novel resistive computing architectures such as Computation-In-Memory (CIM) [11–13], Pinatubo [15], PLiM [14], AC-DIMM [16,43,44]. Typically, these architectures either move the computations into memory arrays [11–14] or embed computing components into the peripheral circuit of the memories [15,16,43,44]. As a result, they could improve the delay and energy performance by at least 10x [11,14,15]. In addition, the non-volatile memristors empower non-volatile computing units [45], such as non-volatile FPGAs [46,47] and processors [48], which can be used in low-power and energy-harvesting systems [48]. However, some crucial topics are still open for research. First, the architecture design methodology has been changed completely. The conventional architectures design the applications, the instruction set, micro-architecture and device technology separately. In contrast, the novel resistive computing architectures need to consider all above factors together to maximize the computing efficiency. Therefore, we need to develop an appropriate design methodology. Second, exploring the details of the architectures at different levels (i.e., macro, micro and nano) and how the different choices can impact the overall performance are still open questions. Third, a relevant instruction set and communication infrastructure should be explored. Finally, design exploration tools (e.g., system simulators) and benchmark suits should be developed in order to evaluate the performance of resistive computing architectures.

Compiler: As resistive computing is still in an infancy stage, only the limited work has been published with respect to the compiler [49–51]. In [50], a domain-specific language is modified to create an appropriate programming interface. In [49,51], the authors proposed a compiler based on arithmetic skeletons to simplify the scheduling process during the compilation; an arithmetic skeleton is an implementation template for a specific class of algorithms. However, the compilers for resistive computing still need efficient programming language interfaces and strong backends that explore the memristor technology fully. In addition, such compilers need to maximally extract parallelism, intelligently map tasks and data onto the massive processing and storage resources, and synchronize the communication between them.

Application: Preliminary results of resistive computing architectures have shown significant performance improvements compared to today's architectures for specific applications, such as genomics in diagnosing/treating diseases, scientific computing, database query, graph processing [11,13,15]. These are applications that benefit from the massive parallelism, ultra-large data storage and energy-efficient computation provided by resistive computing architectures. In addition, as resistive computing architectures can process and transfer data directly in the main memory, data can be processed directly in the main memory without the need to pass it through the whole memory hierarchy to the processor. Although several applications (e.g., DNA sequencing, database query) bene-

fit from resistive computing architectures, it is still crucial to develop novel algorithms that maximize the potential of resistive computing architectures and further improve the overall performance. In addition, it is also necessary to characterize these applications and identify the properties suitable for resistive computing architectures.

1.3. RESEARCH TOPICS

Many challenges described in Section 1.2 still need to be addressed. The research that is carried out in this thesis focuses on two aspects (see also the shadowed boxes in Fig. 1.2). They are logic design (including primitive logic gates, interconnect design, and circuit design and synthesis flow) and architecture (i.e., non-Von Neumann architecture).

PRIMITIVE LOGIC GATE

As logic gates are the fundamental components in logic circuits, it is crucial that they work efficiently. In this thesis, we focus on the exploration of novel logic design styles. These logic styles should be easily integrated into memory arrays and therefore enable the promising resistive computing paradigms. In addition, their resilience against the device variations need to be investigated.

INTERCONNECT DESIGN

To build a complex logic circuit, individual logic gates should be connected with each other. Therefore, appropriate interconnect schemes must be developed. This thesis explores possible interconnect schemes and implementation methods; e.g., using only memristor crossbar, using the peripheral circuits, etc.

CIRCUIT DESIGN AND SYNTHESIS FLOW

In order to use the above gates and interconnect to build large and complex designs, it is critical to explore different design methodologies for both application specific integrated circuits (ASIC) and field programmable gate arrays (FPGA). For the ASICs, solutions of mapping the logic gates onto memristor crossbar and subsequently place-and-route them have to be explored. For the FPGAs, it is crucial to implement look-up tables using different memristor logic styles. In order to benchmark and evaluate the performance of the design methodologies, synthesis flows and performance estimation models for both ASIC and FPGA should be suggested.

NON-VON NEUMANN ARCHITECTURE

As today's von Neuman architecture is suffering from the memory wall, power wall, and ILP wall, we need novel non-Von Neumann architectures. Resistive computing architectures are a promising candidate as they use memristors as both logic and memory and hence reduces the memory access and energy consumption. The potential of such architectures should be analyzed first. Thereafter, the details of such architectures in different levels (i.e., macro, micro and nano) and the suitable applications should be explored and characterized.

1

1.4. CONTRIBUTIONS

The contributions of this dissertation are directly related to the research topics presented in the previous section.

1.4.1. PRIMITIVE LOGIC GATE

We study the existing logic design styles and propose a novel one. With respect to this research topic, the main contributions are as follows:

1. Complete logic gate space exploration for Snider logic [52]. In order to explore the gate space of Snider logic, several gate parameters are extracted from the primitive gates such as circuit structure, fan-in, fan-out, etc. The logic gate space is explored via an exhaustive search of all the gate parameter combinations. Several new gates can be implemented such as AND, NOR, etc.

2. Novel scouting logic, which performs logic operations by modifying standard memory read operations in [31]. Scouting logic implements OR, AND and XOR operations. Instead of reading a single memory cell at a time, scouting logic activates the two inputs of the gate simultaneously. To perform different operations, we only need to use different current references. We modified the standard sense amplifier to a reconfigurable one to support the required functions.

3. Robustness analysis of both Snider [53] and scouting [31] logic gates. For Snider logic gates, a set of proper constraints are formulated to guarantee correct functionality of logic gates (e.g., AND). Its accuracy is evaluated while considering considering the device variability, parasitic resistance and capacitance of nanowires, sneak path currents, and different memristor models. For scouting logic gates, the impact of the device variability is investigated. Subsequently, a variation-resilient design methodology is proposed to select appropriate resistance values for a given failure rate.

1.4.2. INTERCONNECT DESIGN

We explore different approaches to implement interconnects for memristor-based logic design. With respect to this research topic, the main contributions are as follows extracted:

1. Exploration and comparison of three methods to implement general interconnect schemes [54]. First, we only use memristor crossbar to build the interconnect, this interconnect needs to use copy operations within crossbar. The second method is using the peripheral circuit. The peripheral circuit first read out the data from the source and then write the data to the destination. The final method is using some pass transistors to directly connect the source and destination. My work mainly contributes to the interconnects using only memristor crossbars.

2. Crossbar based interconnects to support both intra-tile and inter-tile communication [55]. Based on copy operations, an intra-tile interconnect network for general logic functions is proposed; this interconnect can transfer data between building blocks. In addition, we explore a method to use intra-tile interconnect to solve

specific algebraic problems (e.g., matrix transpose). To implement inter-tile communication, 2D bus is proposed, which enables both horizontal and vertical transmission between tiles (i.e., processing elements).

1.4.3. CIRCUIT DESIGN AND SYNTHESIS FLOW

We develop methodologies to design ASICs and FPGAs. With respect to this research topic, the main contributions are as follows:

1. Design of large scale ASICs [25,56]. First, the place-and-route approaches for large scale ASICs are explored. Thereafter, a peripheral circuit design is proposed, which is used to control the memristor crossbar. Subsequently, an evaluation model is proposed while considering both the memristor crossbar and CMOS controller. Finally, we present a synthesis flow by modifying flows for conventional CMOS logic.

2. Design of FPGAs [47]. First, two different logic design styles are utilized to implement look-up tables. Subsequently, an evaluation model is proposed while considering both the memristor and CMOS parts. Finally, a synthesis flow is developed by adapting existing flows for CMOS based FPGAs .

1.4.4. NON-VON NEUMANN ARCHITECTURE

We explore a non-von Neumann architecture, Computation-In-Memory (CIM Architecture), for specific data-intensive applications. With respect to this research topic, the main contribution is the following:

Development of CIM architecture [11]. CIM architecture is based on the integration of storage and computation in the same physical location. It has a significant potential in solving data-intensive problems (e.g., parallel addition, DNA sequencing) than today's computer architectures in terms of computation efficiency, solving the communication bottleneck, reducing the leakage currents, etc. My major work is implementing related memristor logic circuits and estimating their performance in terms of area, delay and energy.

1.5. THESIS ORGANIZATION

The remainder of this dissertation is organized as follows.

Chapter 2 describes the background on memristor technology (i.e., its brief history and working principle) and its potential applied to memory, logic and resistive computing paradigms (e.g., neuromorphic processing, computation-in-memory).

Chapter 3 discusses the contributions of this dissertation with respect to primitive logic gate design. It presents the proposed logic gates and studies their robustness.

Chapter 4 discusses the contributions of this dissertation with respect to interconnect design. It presents how to design interconnects to connect individual logic gates and ex-

plores the possible implementation methods.

Chapter 5 discusses the contributions of this dissertation with respect to circuit design and synthesis flow. It presents place-and-route approaches for logic gates in ASICs targeting memristor crossbar. Thereafter, it presents FPGA implementations using memristor-based logic circuits. In addition, their synthesis flow and evaluation model are presented.

Chapter 6 describes the contributions of this dissertation with respect to non-von Neumann architectures. We propose Computation-In-Memory (CIM) architecture for specific data-intensive applications (e.g., parallel adder and DNA sequencing); it is based on the capability of memristor technology to integrate both storage and computation in the same physical location.

Chapter 7 concludes this dissertation and discusses the future work.

2

BACKGROUND ON MEMRISTIVE DEVICE AND ITS POTENTIAL

This chapter describes the fundamentals of the memristive device and its potential. First, it briefly provides an overview of the memristive device, including its history and major properties. Thereafter, it describes next-generation non-volatile memories, including Redox Memories, Phase Change Memories, Electrostatic/Electronic Effects Memories. Next, it overviews and compares memristor-based logic design styles (e.g., Snider Logic, Stateful Logic, Memristor Ratioed Logic). Subsequently, emerging computing paradigms are presented, including the computation-in-memory architecture and a neuromorphic architecture. Finally, this chapter highlights major challenges for technology, memory, logic circuits and computing paradigms.

The content of this chapter consists of the following research article:

1. *H.A. Du Nguyen, J. Yu,* ***L. Xie****, M. Taouil, S. Hamdioui, D. Fey, Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Abu Dhabi, UAE, October, 2017, pp. 1-10*

Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann

2

H.A. Du Nguyen*, Jintao Yu*, Lei Xie*, Mottaqiallah Taouil*, Said Hamdioui*, Dietmar Fey†
*Computer Engineering, Delft University of Technology, Delft, the Netherlands
S.Hamdioui@tudelft.nl
† Computer Architecture, University of Erlangen-Nrnberg, Erlangen, Germany
dietmar.fey@informatik.uni-erlangen.de

***Abstract*—Traditional CMOS technology and its continuous down-scaling have been the driving force to improve performance of existing computer architectures. Today, however, both technology and computer architectures are facing challenges that make them incapable of delivering the growing computing performance requirement at pre-defined constraints. This forces the exploration of both novel architectures and technologies; not only to maintain the economic profit of technology scaling, but also to enable the computing architecture solutions for big-data and data-intensive applications. This paper discusses the emerging memristive device as a complement (or an alternative) to CMOS devices and shows how such devices enable novel computing paradigms that will solve the challenges of today's architectures for certain applications. The paper covers not only the potential of memristor devices in enabling novel memory technologies, logic design styles, and arithmetic operations, but also their potential in enabling in-memory computing and neuromorphic computing.**

I. INTRODUCTION

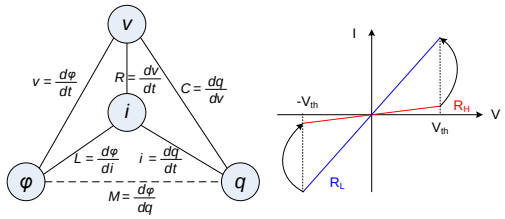
Today's and emerging applications including internet-of-things (IoT) and big data applications are extremely demanding in terms of storage and computing performance. Such world-changing applications will not only impact all aspects of our daily life, but also change a lot in the IC and computer manufacture industry. Emerging applications require computing performance which was typical of supercomputers a few years ago, but with constraints on size, power consumption and guaranteed response time which are typical of the embedded applications [1,2]. Both current device technologies and computer architectures are encountering significant challenges that make them incapable of providing the required functionalities and properties.

Nanoscale CMOS technology is facing three walls [2]: (1) the reliability wall as technology scaling leads to increased failure rate and reduced device lifetime [2], (2) the leakage wall as static power dominates and might be even larger than dynamic power at more advanced technology nodes (due to volatile technology and decreasing supply voltage) [3]; (3) the cost wall as the cost per transistor via pure geometric scaling of process technology is plateauing [4]. These walls have led to the slowdown of the CMOS scaling. On top of that, today's computer architectures are facing the three well-known walls [5]: (1) the memory wall due to the growing gap between processor and memory speeds, and the limited memory bandwidth making the memory access as the killer of performance and energy consumption for data-intensive applications; e.g. big-data; (2) the Instruction Level parallelism (ILP) wall due to the complexity of extracting sufficient parallelism to keep all cores running; (3) the power wall as the practical power limit for cooling is reached, which leads to no further increase of CPU clock frequency. In order for computing systems to continue delivering required performance and sustaining profits for the near future, alternative computing architectures have to be explored in the light of emerging device technologies. Resistive computing, neuromorphic computing and quantum computing are some candidates for the next-generation computing paradigms, while memristor devices, quantum dots, spin-wave devices are couple of emerging device technologies [6]. Among these technologies, memristor is a promising candidate to complement and/or replace traditional CMOS (at least for some applications) due to many advantages such as near-zero standby power, high device scalability, high integration density, and CMOS process compatibility [7,8]. Therefore, it provides significant potential to implement high density memories [9–11], different logic design styles [12–16], and consequently enabling new computing paradigms [17–21].

This paper will comprehensively explore the potential of memristors in building logic functions, memories, arithmetic operations, and novel computer architectures. Section I briefly describes the history and characteristics of memristive devices. Section II and III overview the logic design styles and non-volatile memories based on memristive devices, respectively. Section IV shows how the unique properties of memristor devices enable the concept of neuromorphic and emerging computation-in-memory architecture. Section V highlights the major challenges for memristive device based computing, followed by a conclusion of this paper.

II. MEMRISTIVE DEVICES: WHAT ARE THEY?

Memristive device, better known as memristor, is the fourth fundamental two-terminal element, next to the resistor, capacitor, and inductor. It was initially proposed in 1971 by the circuit theorist Leon Chua [22]. He noticed that there was still a missing relationship between flux and charge as shown by the dashed line in Fig. 1(a). Theoretically, a memristive device is a passive element that maintains a relationship between the time integrals of current and voltage across a two-terminal element, while considering the internal state variable of the device.



(a) The four fundamental elements (b) Pinched hysteresis loop

Fig. 1: Stateful Logic

Hence, a memristor can be expressed either by a function of charge q or flux ϕ. An important fingerprint of a memristor is the pinched hysteresis loop current-voltage characteristic as illustrated in Fig. 1(b). It shows that memristive devices have two stable states: high R_{H} and low R_{L} resistive states. When the voltage across the memristive device is greater than the absolute value of its threshold voltage (i.e., V_{th}), then it switches from one resistive state to another. Secondly, it has the ability to remember its history (i.e., the internal state).

After a silent period of more than 30 years, memristive device became renowned in 2008 when the first physical memristor device was fabricated by HP Lab [23]. HP built a metal-insulator-metal device using a titanium oxide as a insulator sandwiched by two metal electrodes. They successfully identified the memristive behaviour over its two-terminal node as described by Leon Chua. The device tunes its resistance by controlling positive charged oxygen vacancies in the insulator layer by applying different voltages. After the first memristive device was manufactured, many memristor devices based on different type of materials have been proposed such as HfO_x, TaO_x, SiO_x [7,8].

III. MEMRISTIVE DEVICES FOR LOGIC

This section first classifies existing memristor-based logic design styles. Thereafter, it briefly describes examples of each class. Finally, it qualitatively compares them.

A. Classification

Multiple logic design styles have been proposed [12–16,24–27]. We divide them into several classes using the following criteria:

- **Input Data Representation** indicates whether the input data is represented by a voltage or resistance.
- **Output Data Representation** indicates whether the output data is represented by a voltage or resistance.
- **Processing Elements** indicates whether the data is *processed* based on memristors only or by using a hybrid cmos/memristor combination. Obviously the *control* of the memristors is always done using CMOS circuits.

Fig. 2 shows the classification result; there are eight classes in total. Each class is named based on the input and output representation signals, and the processing element. For instance,

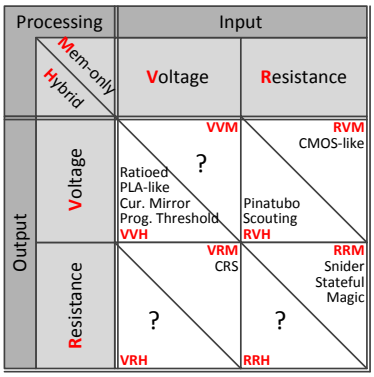


Fig. 2: Classification of Memristor-Based Logic Design Styles.

scouting logic is located in the RVH class where R indicates the input data representation, V the output data representation and H hybrid CMOS/memristor processing. The classification clearly shows that the existing logic designs fit in five defined classes, and that three classes are potentially not explored yet.

- **VVH**: Memristor ratioed logic [24], PLA-like [12], current mirror based threshold logic [13], and programmable threshold logic [25] belong to this class. They use a voltage to represent both input and output data and CMOS gates (e.g., inverter [12,13,24] and D Flip-Flop [25]) as a threshold function (and inverter). The memristors are used as either configuration switches [12,24] or input weights [13,25].
- **RVH**: Pinatubo [28] and Scouting logic [27] are the work published in this class. They use a resistance to represent the input data and a voltage to represent the output data. Both logic styles perform logic operations by modifying memory read operations.
- **RVM**: CMOS-like logic [26] is the only existing work in this class. It uses a resistance to represent the input data and a voltage to represent the output data. It replaces MOSFETs in the pull-up and -down network of the conventional CMOS logic with memristors.
- **VRM**: Complementary Resistive Switching (CRS) logic [14] is the only published work in this class. It uses a voltage to represent the input data and a resistance to represent the output data. CRS logic performs logic operations by modifying memory write operations. In addition, You et al. extended the existing CRS logic gates with other Boolean logic gates which requires also fewer execution steps [29].
- **RRM** Snider [15] and stateful [16] logic belong to this class. They use a resistance to represent both the input and output data. They perform logic operations by using memristors as voltage dividers which conditionally switch the output memristors. Lehtonen et al. [30] extended stateful logic to support more types of logic operations (e.g., AND-IMP and OR-IMP). Kvatinsky et al. [31] and

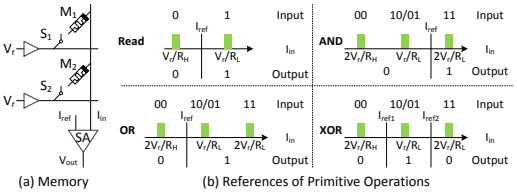


Fig. 3: Scouting Logic

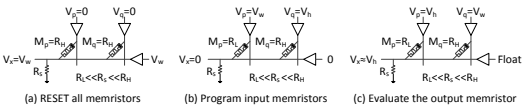


Fig. 4: Stateful Logic

Xie et al. [32] extended Snider logic to support more types of logic operations (e.g., AND and OR).

In the remainder of this section, the working principle of two logic design styles will be given as examples since they are the most popular candidates to implement resistive computing systems. Finally, a comparison between the state-of-the-art will be provided.

B. RVH: Scouting Logic

As Pinatubo and scouting logic share the same idea, we use scouting logic as an example using different circuit implementations. Scouting logic [27] supports the AND, OR and XOR logic operations. Scouting logic uses resistances R_H and R_L to represent its logic inputs 0 and 1, respectively; it uses voltages V_{dd} and *GND* to represent its logic output 1 and 0, respectively.

Scouting logic is inspired by memory read operations. Typically when a cell is read, say Memristor M_1 of Fig. 3(a), a read voltage V_r is applied to its row and the switch S_1 is activated. Subsequently, a current I_{in} will flow through the bit line to the input of the sense amplifier (SA). This current is compared to the reference current I_{ref}. If I_{in} is greater than I_{ref} (i.e., when M_1 is R_L state), the output of the SA changes to V_{dd} (logic 1). Similarly, when M_1 is R_H state, I_{in}<I_{ref} and subsequently the output changes to logic 0. For proper operations, I_{ref} should be fixed between high and low currents of Fig. 3(b). Instead of reading a single memristor at a time, scouting logic activates the two inputs of the gate simultaneously (e.g., M_1 and M_2 in Fig. 3(a)). As a result, the input current to the sense amplifier is determined by the equivalent input resistance ($M_1//M_2$). This resistance results in three possible values: $\frac{R_L}{2}$, $\frac{R_H}{2}$ and $R_L//R_H \approx R_L$. Hence, the input current I_{in} can have only three values. By changing the value of I_{ref} different gates can be realized.

For example, to implement an OR gate I_{ref} should be set between $\frac{2V_r}{R_H}$ and $\frac{V_r}{R_L}$ as depicted in Fig. 3(b)). When the inputs are $p = 0$ and $q = 1$, the input current I_{in} to the sense amplifier is around $\frac{V_r}{R_L}$. As $\frac{2V_r}{R_H}$<I_{ref}<$\frac{V_r}{R_L}$, $I_{in} > I_{ref}$ and the output voltage V_{out} is V_{dd}. The AND and XOR operations work in a similar way. Note that the XOR gate needs two references which is not shown in Fig. 3(a). More details on the sense amplifier can be found in [27].

C. RRM: Stateful Logic

Stateful logic [16] supports material implication (IMP) as primitive logic operation. The IMP operation is denoted by Eq. 1.

$$\text{IMP: } q' = p \rightarrow q = \bar{p} + q \tag{1}$$

Here p and q are inputs while q' is the output. Stateful logic uses R_H and R_L represent logic 0 and 1, respectively; both for the inputs and outputs. An IMP gate consists of two memristors (i.e., M_p and M_q) and a resistor R_s ($R_L \ll R_s \ll R_H$). M_p is only used for the input p while M_q is used both for the input q and output q'. To perform the operation, control voltages V_h and V_w are applied to M_p and M_q, respectively; the control voltages typically satisfy the relationship: $0<V_h=\frac{V_w}{2}<V_{th}<V_w<2V_{th}$.

To illustrate the working principle of stateful logic, an example of an IMP gate is given for the inputs $p = 1$ and $q = 0$, as shown in Fig. 4. It consists of three steps. First, all the memristors are reset to R_H by applying voltages $V_p = V_q =$*GND* and $V_x = V_w$ (see Fig. 4(a)). Second, M_p is programmed to R_L ($p = 1$) by applying voltages $V_p = V_w$, $V_q = V_h$ and $V_x = 0$ (see Fig. 4(b)). V_h is used to prevent M_q from undesired switching. Finally, the IMP gate is evaluated by applying voltages $V_p = V_h$, $V_q = V_w$ and keeping the row floating (see Fig. 4(c)). Therefore, $V_x \approx V_h$ ($R_L \ll R_s \ll R_H$) and the voltage across M_q is $V_q - V_x \approx V_w - V_h < V_{th}$. As a result, M_q stays in R_H. More details and the latest progress can be found in [16,30,33].

D. Comparison

We use the following metrics to qualitatively compare the existing memristor logic design styles.

- **Array Compatibility** indicates whether the logic style is compatible with normal 1R and/or 1T1R memory arrays or not.
- **CMOS Controller Requirement** indicates whether the logic style needs a CMOS circuit to control it or not.
- **Nonvolatility** indicates whether the logic style can store the data when it is powered off or not.
- **Area** indicates how area-efficient the logic style is to perform operations.
- **Speed** indicates how fast the logic style is to perform operations.
- **Energy Consumption** indicates how energy-efficient the logic style is to perform operations.
- **Scalability** indicates how well the logic style can be scaled to implement more complex circuits.
- **Robustness** indicates how robust the logic style is to be resilient against the unrelaible CMOS and memristor technology.

TABLE I: Comparison Between Existing Logic Styles

Style	Class	Array	Control	NV	Speed	Area	Energy	Scalability	Robustness
Memristor ratioed logic	VVH	No	No	No	+	++	++	++	+
PLA-like memristor logic	VVH	No	No	No	+	++	++	++	++
Current mirror threshold logic	VVH	No	No	No	+	++	++	++	++
Programmable threshold logic	VVH	No	No	No	+	++	++	++	++
Pinatubo / Scouting logic	RVH	Yes	Yes	Yes	+	+	++	+	+
CMOS-like logic	RVM	No	Yes	Yes	-	-	-	-	+
CRS logic	VRM	Yes	Yes	Yes	-	-	-	-	-
Snider logic	RRM	Yes	Yes	Yes	-	-	-	-	-
Stateful logic	RRM	Yes	Yes	Yes	-	-	-	-	-

Table I shows the comparison result. We can draw the following conclusions with respect to the metrics.

- **Array Compatibility**: Design styles of RVH, VRM and RRM are compatible with memory arrays. CMOS like memristor logic is not compatible with memory arrays due to its irregular topology. Design styles of VVH are not compatible with 1R/1T1R array as they need to add CMOS inverters or D flip-flops to memory arrays. Note that array compatibility is an important requirement to implement resistive computing systems.
- **CMOS Controller Requirement**: The logic styles of VVH do not need additional CMOS control units as their inputs and outputs are voltage based. In contrast, other logic styles need to transduce the data between voltages and resistances, and also need the controller to control each step during execution. Note that several logic design styles require multiple execution steps.
- **Nonvolatility**: Only the design styles of VVH are volatile, as both their inputs and outputs are represented by voltages. In contrast, other logic styles have their input and/or output represented by resistances, and thus are nonvolatile.
- **Speed**: The design styles of VVH and RVH are faster as they can finish logic operations in a single step. In contrast, other logic design styles are slower as they need multiple steps.
- **Area**: Design styles of VVH require smaller area as they do not need CMOS controllers. In contrast, other design styles require larger area as they need CMOS controllers. In addition, Pinatubo/Scouting logic needs a simpler controller as it only needs a single step instead of multiple [27].
- **Energy Consumption**: Three main factors impact on the energy consumption; they are controller necessity, nonvolatiltiy and speed. Design styles of VVH do not need CMOS controllers and they are fast, and hence they are likely not to consume a lot energy to perform logic operations. Design styles of RVH are nonvolatile and fast, and hence they are likely to consume less energy to perform logic operations. In contrast, the other design styles possibly need more energy as they need complex controllers and longer time to perform logic operations.
- **Scalability**: Controller necessity impacts on the scalability. Design styles of VVH are the easiest to scale up as they do not need CMOS controllers. Design styles of RVH are easier to scale as they need a simpler controller. In contrast, the other design styles are hard to scale up as they need complex controllers.
- **Robustness**: Controller necessity impacts on the robustness as many transistors are involved by controllers. In addition, design styles are more reliable if the memristors do not need to switch during logic operations. This is because memristor devices suffer from cycle-to-cycle variation [2]. Except memristor ratioed logic, design styles of VVH are likely to be most robust as they need no CMOS controllers and memristor switching. Design styles of RVH and RVM are more reliable than other styles as they do not need to switch memristors during logic operations.

Overall, in order to implement the resistive computing architectures, design styles of RVM , VRM, and RRM are very suitable due to their array compatibility. Among them, scouting logic is the most promising candidate due to its good performance in the remaining aspects. In addition, the design styles of VVH and RVM are possible alternatives to replace CMOS logic.

IV. MEMRISTIVE DEVICES FOR MEMORIES

Many non-volatile memory elements have been proposed such as phase-change-memories (PCMs), spin-torque-transfer magnetic RAMs (STT-MRAMs), and resistive RAMs (ReRAMs). A very good introduction into the the topic of memristive memory and the ReRAM technology is given in the first two chapters [34], [35], in the book *Resistive Switching*, edited by Ielmini and Waser [36].

Each of these device classes shows a more or less different technology and working principle causing different benefits and drawbacks what itself leads to different appropriate use scenarios of these devices. In the following we will briefly show an overview of these memristive devices used as memories.

PCMs are based on the use of calcogenide materials which can be switched between an amorphous and a crystalline state.

This is realised by heating up a conductive rod reaching through the calcogenide material with a high write current. The two states show different behaviours in their electric resistance when the current is flowing through such a device. If the calcogenide is crystalline, the whole device is in the low resistance state (LRS), in contrast it is in the high resistance state (HRS) if the calcogenide is amorphous. Furthermore, it is also possible to adjust intermediate states which are located between the two extremes, the LRS and the HRS. This possibility leads us to the first benefit of such PCM devices, namely its feasible multi-level cell operation. Additionally, PCMs offer a quite mature technology and show a good compatibility (MLC) with CMOS. PCMs have more than 10^9 an endurance comparable to ReRAMs which have the best endurance of current non-volatile memristive devices. The endurance corresponds to the maximum number of possible switching cycles up to the moment, in which the device does not work anymore. On the other side there are some challenges in the controlling of the switching process. This refers to the necessary high write circuits, a 10x slower switching speed than ReRAMs due to the slow crystalline process, and the resistance drift in the amorphous state that has to be compensated on circuit level.

STT-RAMs are based on a parallel and anti-parallel configuration of a stack of ferromagnetic layers forming a magnetic tunnel junction (MTJ) structure. The magnetization at the terminals of the MTJ stack is on one side fixed, therefore this side is denoted as a fixed layer, whereas on the opposite side the so-called free layer is located, which can be switched between two magnetization directions. If both layers are in parallel to each other, the electrons with opposite orientation spin-polarized can pass with a high probability through the stack. Therefore in this case the device is in HRS. In contrast, if the two layers are polarized anti-parallel to each other the probability that an electron can pass both layers is low, since the electron will always meet a layer with different polarization to its own one independent in which direction the electron is spin-polarized. Therefore in this case the device is in a HRS. The benefits of such technology is the fast switching and its relatively mature technology even if it is a challenge to make it compatible with CMOS because the MTJ stack can consist of more than ten layers of not easy to handle ferromagnetic materials, e.g. CoFeB or MgO. Nevertheless, due to its low energy efficient features STT-MRAM technology is strongly discussed to use them in last-level caches.

The ReRAM technology can be subdivided in three different approaches which all exploit nanoionic switching mechanisms. These three approaches are typified either as electrochemical memory (ECM), valence change memory (VCM) (see Fig. 5), or thermochemical memory (TCM) which are using different ionic mechanisms to generate different resistances. In TCMs and ECMs a so-called filamentary structure is used to build up and down small metallic bridges by redoxation and oxidation processes in ionized material layers consisting of e.g. TiO_2 or HfO_2 which are entangled between two metal plates as terminals. TCMs are unipolar, i.e. the same voltage is applied

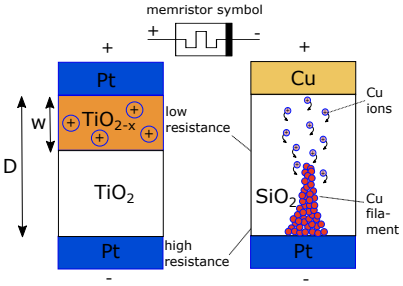


Fig. 5: ECM (left) and VCM (right) ReRAMS (ECM).

to the poles and a filament with low resistance characteristic is growing from both sides. In contrast to that, in ECM two opposite voltages are applied to the terminals which are normally composed of different metals. By this bipolar control mechanisms voltage and reversed voltage signals are used to build up the metallic filament by a redox transitions and to dissolve it again by launching local oxidation processes.

In VCMs not only a filament but also a complete metallic layer or an area interface is built up and dissolved by the exchange of ions. VCMs are also bipolar devices and they correspond to the technique that was used in the memristors of Hewlett Packard [37]. Due to the focus on ion motion as underlying switching process much more localized structures in the nanometer range sized cells, e.g. 10x10 nm^1 or even less, can be realized offering good scalability. A large HRS / LRS ratio makes the interfacing to resistance evaluating CMOS circuits easier. ReRAMs are further characterized by fast switching in the ns range, but even 100 ps have been demonstrated. This characteristic is given due to the small distances the ions have to move and the high electrical field forces that occur in the nanoscale active region causing a so-called Joule heating what for its part further increases the ion mobility. A further advantage is the good compatibility of ReRAMs with CMOS manufacturing processes even if 3D integration of PCMs is a little bit easier since PCMs need only a unipolar selection device compared to the bipolar ReRAMs switching. The endurance, which is may be the most important feature for memristive elements concerning their use in computing circuits either as memory or as switching element, is reported very different in literature. One can find values of 10^6 cycles up to more than 10^{12} cycles. The power comsumption is in the pf range, which makes ReRAMs a good candidate for an use in embedded applications. For example, in 2013 Panasonic is the first semiconductor manufacturer who integrated ReRAM into their microcontroller for storing firmware [38].

V. MEMRISTIVE DEVICES FOR ARITHMETIC

This section lays a focus on how memristive devices can be used to realise new computing concepts for arithmetic circuits. In particular, the presented concepts will exploit qualitative benefits of memristive devices that can not be so easily realised

with pure SRAM or DRAM memory cells. This will be on the digital side the multi-level cell (MLC) capability which can be used for new ternary computing concepts like e.g. ternary adders. Ternary content-addressable memory (TCAM) is another architecture in which memristive devices are used for ternary computing schemes [39]. However, TCAM does not exploit the MLC feature. It uses two memristors for a storing a logical 1 and 0, and an additional third memristor for the realisation of the *don't care* state, which is essential for a CAM. Therefore, the rest of this section will only focus on the ternary computing.

Actually, ternary computing schemes for arithmetic operations like addition, subtraction, multiplication and division are long known. First mathematical investigations go back to the 17^{th} century. Two newer ground breaking work was done by Avizienis [40] and Parhami [41]. The first work shown that carry-free additions can be realised by using so-called signed-digit numbers to a base $r \geq 3$. It means that numbers are not presented in the usual sign magnitude presentation, like e.g. in one's or two's complement, but each digit can also have negative values. Then, independent of the operand's word length an addition can be carried out in $O(1)$ instead of $O(N)$ or $O(log(n))$ which is unavoidable if pure binary numbers and at the same time a reasonable number of integrated Boolean gates is utilized. In 1988 Parhami [41] presented a solution in which also a base $r = 2$ can be used to make the realisation with digital electronics possible.

The question remains why such ternary concepts were not used in the last decades in integrated microprocessors if their benefits concerning the run time of arithmetic operations are obvious. One answer to that question is that no CMOS compatible storage device was available that could store three states. This had led to a situation that the complete register files, the caches and even the data segments of the main memory had to be doubled by two SRAM or DRAM cells to store three states. With the emerging of MLC-capable memristive devices this situation has changed. The idea to use MLC memristive devices for ternary adders was first published in [42]. The first technical solution using MLC ReRAMs for redundant arithmetic operations is shown in [43]. The clear qualitative benefit over SRAM and DRAM memory for ternary arithmetic can be exploited in two directions. First, MLC based memristive devices can be used as ternary memory in digital CMOS circuits or, second, in pure in-memory computing circuits in which a well-directed state transfer between the three states in one memristive device is induced according to the compute rules of ternary computing.

The way how a ternary addition works is explained by means of the example shown in the Table II. The basic idea to avoid a carry transfer over more than two digits is that in step 1 only 0, -1, or -2 is used for the result of a digit in the intermediate sum z; whereas in the so-called transfer vector t only 0 or a positive 1 is used. This avoids a further generation of a carry. The rules of math have to be observed, i.e. $0 + 1$ yields -1 for z_i and 1 for t_{i+1}. The second step is necessary to get rid of the -2 in digit z_1. Now, -2 turns to -1 in digit t'_2.

TABLE II: Addition of Two Ternary Numbers.

x =	(0	0	-1	0)$_2$	= $(-2)_{10}$
+y =	(0	1	-1	0)$_2$	= $(+2)_{10}$
step 1:					
	0	-1	-2	0	= z
0	1	0	0	0	= t
step 2:					
0	1	1	0	0	= z'
0	-1	-1	0	0	= t'
step 3:					
0	0	0	0	0	= $s = 0$

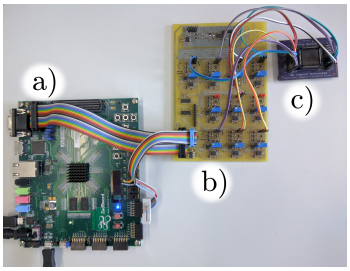


Fig. 6: Prototyping Platform for Memristive Ternary Adder.

In general, by repeatedly applying the rules of addition, only 0s and 1s are generated in the vector z' while only 0s and -1s in vector t'. Then, no situation can occur that two positive 1s or two negative 1s will meet at the same digit position and no carry can occur. Therefore the addition requires exactly three steps independent of the operands' word length.

Details about the complete Boolean logic for the ternary compute steps and an extensive comparison with other possible ternary representations concerning a solution with MLC capable memristive devices can be found in [44].

The memristive ternary adder was realized as a first prototype using discrete electronic devices (Fig. 6) consisting of an FPGA board (a) that implements the Boolean logic for the ternary adder, a device from BioInspired Inc. (c) to provide the memristors, and an interface card (b) designed by our own which realises the communication between the FPGA and the memristor device via ADC and DAC functions. More details in the set up can be found in [45].

Fig. 7 shows a measurement of the memristor device used as ternary storage. There are two sets of five measurement curves to see. Each curve shows the current running through the memristor by reading the memristor with an applied low voltage after we wrote a memristor in subsequently increased voltage steps of $\delta V = 0.5\ V$. This was done for two different compliant current limits of 10 μA and 40 μA. Even though the five curves for a certain current compliance level scatters widely, we still could clearly distinguish the three desired states.

Fig. 8 shows a screen display of the program that controls

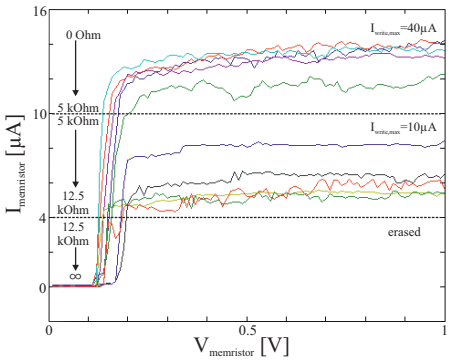


Fig. 7: Determine the Resistance of a Memristor.

```
Datei  Bearbeiten  Ansicht  Suchen  Terminal  Hilfe
Memristor test environment
------------------------------
1) Read all memristors
2) Write register 0
3) Write register 1
4) Add SD
5) Add SD R0+R1->R1
------------------------------
6) All =0
7) All =1
8) All =-1
9) Show threshold values

|MEMR    |VALUE   |ADCLEVEL|
+--------+--------+--------+
|       0|       1|   16674|
|       1|       1|   24492|
|       2|       1|   15681|
|       3|       0|   65535|
|       4|      -1|   61994|
|       5|      -1|   49575|
|       6|       0|   65535|
|       7|       0|   65535|

Register 0
+----+----+----+----+----+
| 00 | 00 |  1 |  1 |  1 |
+----+----+----+----+----+

Register 1
+----+----+----+----+----+
|  0 |  0 | -1 | -1 |  0 |
+----+----+----+----+----+
```

Fig. 8: Testing Program for a Memristive Ternary Adder.

carried out experiments. Multiple of the in all 20 memristor cells in the memristor device (Fig. 6c) were addressed as ternary registers. Their content was read in the FPGA. Then, the new ternary result was calculated there and written back to the ternary memristor device.

VI. MEMRISTIVE DEVICES FOR RESISTIVE COMPUTING

Resistive computing enabled by memristive technology has introduced new opportunities to renovate existing computing paradigms for embedded and low power computing [46,47], in-memory computing [18,21,48] as well as neuromorphic computing [49,50].

The rest of this section will describe the Computation-in-Memory (CIM) and neuromorphic processing as examples of the emerging resistive computing.

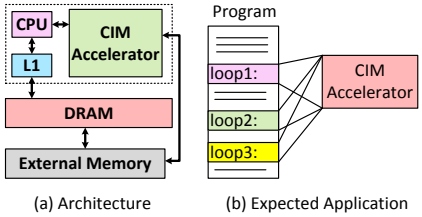


Fig. 9: CIM-based Architecture

A. Computation-in-Memory

One potential in-memory computing architecture is CIM, which was introduced in [18] based on the concept of integrating computation and storage units in a dense memristor crossbar. CIM is implemented as an accelerator (for specific applications) and integrated into conventional architectures to improve overall computer performance [2,18].

The CIM-based architecture consists of a conventional processor, caches, CIM accelerator, main memory DRAM and external memory (as shown in Fig. 9(a)). Similarly as in conventional architectures, the processor fetches, decodes and executes a big data program. However, in conventional architectures, the intensive memory accesses consume (relative to ALU instructions) an enormous amount of energy and significantly degrade the overall performance due to frequent cache misses. As compared to an ALU operation, loading a word from the on-chip SRAM (50x) and off-chip DRAM (6400x) cost much more energy [51,52]. Eliminating this communication will impact the overall performance significantly, especially for data-intensive applications. In order to reduce the data transfers between caches and memories, the CIM accelerator will execute the data-intensive parts of the program locally within the CIM accelerator. Note that the CIM accelerator can perform parallel operations locally on the data stored in the non-volatile memory, hence the memory bottleneck can be significantly reduced. Therefore, the CIM architecture achieves significant improvements in both performance and energy consumption. The performance can be further improved if appropriate applications are mapped on the CIM accelerator.

The potential applications that benefit from CIM accelerator are (big data) applications where communication between processor and memory results in a low performance and high energy consumption. In case the CIM accelerator's capacity is large enough to store the application data, a high level of parallelism can be exploited. In addition, a higher performance can be achieved when different operations are applied to the same data, i.e., data that is not changing frequently; this also benefits the endurance of the non-volatile memory of the CIM accelerator. Last but not least, if the processor provides appropriate instructions to the CIM accelerator ahead of the normal execution time, the CIM accelerator already can start performing its operations while the CPU is simultaneously executing other operations, resulting in overall performance

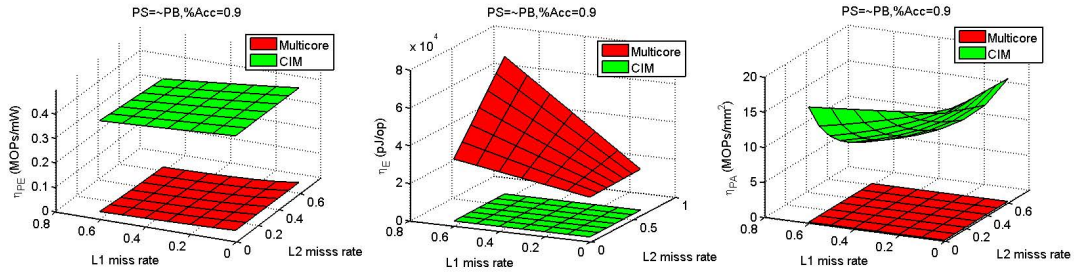


Fig. 10: Evaluation Results for CIM and Multicore Architectures

improvement.

Fig. 9(b) illustrates a program that could be executed efficiently on the CIM accelerator. In this case, multiple invoked loops work on the same large datasets; obviously the data should be initialized on the CIM accelerator. Each time the loop is invoked, the processor sends a request to the CIM accelerator; the latter, performs the requested operations and returns the results to the processor. Examples of such applications are database applications, where multiple queries (each consisting of large loops) are applied to a fixed database. These queries are used to look for specific data patterns in the database.

To illustrate how the CIM-based computer architecture advances the state-of-the-art, its estimated performance will be compared to a multicore-based architecture. The assumptions for the multicore architecture and CIM-based architecture are similar to those in [53]; the multicore architecture consists of 4 cores (ALU only), two levels of caches (32KB L1 and 256KB L2) and 4GB DRAM; the CIM-based architecture consists of one core (ALU only), two levels of caches (32KB L1 and 256KB L2), 2GB DRAM, and a CIM accelerator with a dedicated computing unit and a 2GB memory. The non-accelerated part is executed by the ALU of the conventional processor and the accelerated part by CIM's dedicated computing unit. The memory operations are modeled based on cache miss rates and DRAM access time, similarly as provided in [18,53]. Three metrics are used for the evaluation: (1) performance energy efficiency η_{PE} (defined by MOPs/mW), (2) energy efficiency η_E (defined by pJ/op), and (3) performance area efficiency η_{PA} (defined by MOPs/mm^2).

Fig. 10 shows the results of the evaluation metrics for both architectures. It assumes that 90% of the instructions can be accelerated on CIM. Both architectures execute petabyte problem size. As the organization of the CIM-based architecture preserves the conventional part of a multicore architecture (i.e., CPU, caches, DRAM and external memory), only 10x improvement is obtained with respect to the performance-energy efficiency. However, the CIM-based architecture achieves four orders of magnitude energy efficiency improvement in comparison with the multicore architecture. Furthermore, the proposed architecture is 15x area-efficient than the multicore architecture. In comparison with state-of-the-art, the proposed architecture is capable of realizing significant improvements, despite the high switching latency and low endurance of memristor technology. The improvements are the result of a significant reduction of cache and DRAM accesses and the usage of non-volatile memory. The reduction of memory accesses leads to a lower latency and lower energy consumption, while the non-volatile memory reduces the static power to practically zero.

B. Neuromorphic Processing

Memristive devices related to neuromorphic processing concern the weights that have to be used in artificial neural networks. They are quasi naturally mapped onto different stored resistance levels. Moreover, the neuromorphic processing systems use memristive devices as not only processing elements but also storing elements, which is not possible using CMOS memories. This finally leads to more energy-efficient and smaller circuits as compared to current CMOS solutions.

Actually, the proposals found in literature to use memristive devices for analogue processing came much earlier and have also a much higher number than the proposals for digital solutions. The idea, e.g. for memristive spike-time-dependent plasticity (STDP) networks [54], [55] is to mimic directly the functional behaviour of a neuron. In STDP networks the strength of a link to a cell is determined by the time correlation of incoming signals to a neuron along that link and the output spikes. The shorter the input pulses occur compared to the output spike, the stronger the input links to the neuron are weighted. In contrast, the longer the input signals lay behind the output spike, the weaker the link is adjusted. This process of strengthening or weakening the weight shall be directly mapped onto memristors by increasing or decreasing their resistance depending which voltage polarity is applied to the terminals of a two-terminal memristive device. This direct mapping of an STDP network to an analogue equivalent of the biological cells by an artificial memristor based neuron cells shall emerge new extreme low-energy neuromorphic circuits.

An example how memristive devices are used for neuromorphic processing is a chip based on PCM technology realized by IBM. The chip comprises 64k cells, consisting of 256 axons by 256 dendrites and was demonstrated in

2015 [56]. The update of the artificial synaptic weights uses STDP as an in-situ learning function. Besides this memristor based STDP networks, there are lots of other proposals, e.g. [57], for neural networks to be realised with memristor based crossbar and mesh architectures for cognitive detection and vision applications.

VII. Opportunities and Challenges

A. Opportunities

Memristive devices provide significant opportunities with respect to the following aspects.

- Memristive devices are a promising alternative for the leakage wall that limits CMOS technology scaling.They are non-volatile and require low energy consumption; hence, they can be used to produce memory and logic circuits that have practically zero static power [58–60].
- Memristive devices could also help solving the computer architecture walls. Memristive devices are capable of both storing and computing, which enables new computing paradigms [18,52,61]. Furthermore, massive parallelism can be achieved as the high density of memristive devices facilitate more functional units within the same area [20,53]. In addition, the high energy consumption of today's computers can also be reduced due to the low dynamic energy and zero static energy consumption [53].
- The above benefits (non-volatility, low energy consumption and scalability) provide solutions for emerging applications such as embedded and low power systems for IoT applications [62], big data application and health care applications [18].

B. Challenges

Despite the above opportunities, memristive devices are facing several challenges.

- Memristive devices are still in their infancy stage; hence, there is a lack of libraries with well-optimized memristive designs as well as mature automation tools to speed-up the exploration process.
- Memristive devices suffer from limited endurance which is currently around (10^{12}) [8]. This is insufficient for general purpose computing; therefore, a new computing paradigm or logic style is required to deal with this limited endurance (e.g., high-radix computing [63]). In addition, researchers strongly believe that the endurance will substantially increase and reach 10^{16} [64]
- Memristive devices also face reliability challenges in both manufacturing and design phase. Nonvolatile memory manufacturing has to deal with nonuniform resistance profile across the crossbar array, resistance drift, inherent device-to-device and cycle-to-cycle variations as well as yield issues [2]. Furthermore, the intrinsic variation of memristive devices make it difficult to design robust logic circuits [2,65].
- The integration of memristive devices with CMOS is still an open research question. It seems that this kind of integration is possible as presented in [66]; however, details of stacking memristive layers on top of CMOS are still in research. Moreover, there is limited work that investigates the impact of the CMOS controller on an entire memristive system [53]. Further investigation requires not only the optimization of the CMOS part, but also its efficient use to control the crossbars while maintaining the system scalability.

VIII. Conclusion

Memristive devices provide many opportunities; not only to enable next-generation non-volatile memories (with fast access speed, up to 1TB storage capacity, and multi-level capability), but also to enable novel alternative computing architectures required for emerging applications, such as energy-efficient neuromorphic systems and computation-in-memory architectures. However, there are still many challenges ahead that need to be addressed.

References

[1] EU-Commission, "Next generation computing roadmap." European Union, 2014.

[2] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *DATE*. IEEE, 2017.

[3] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.

[4] H. Jones, "Whitepaper: Semiconductor industry from 2015 to 2025." International Business Strategies, 2015.

[5] J. L. Hennessy *et al.*, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[6] ITRS, "Beyond cmos white paper." ITRS, 2014.

[7] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, pp. 2632–2663, 2009.

[8] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.

[9] R. Fackenthal *et al.*, "A 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 338–339.

[10] M.-F. Chang *et al.*, "Challenges and circuit techniques for energy-efficient on-chip nonvolatile memory using memristive devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 183–193, 2015.

[11] A. Kawahara *et al.*, "An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 178–185, 2013.

[12] J. Borghetti *et al.*, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proceedings of the National Academy of Sciences*, vol. 106, pp. 1699–1703, 2009.

[13] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.

[14] E. Linn *et al.*, "Beyond von neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, p. 305205, 2012.

[15] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, pp. 1165–1172, 2005.

[16] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.

[17] Q. Guo *et al.*, "Ac-dimm: associative computing with stt-mram," *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 189–200, 2013.

[18] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1718–1725.

[19] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *NANOARCH*. IEEE, 2015.

[20] A. Haron *et al.*, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[21] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[22] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, pp. 507–519, 1971.

[23] D. B. Strukov *et al.*, "The missing memristor found," *nature*, vol. 453, p. 80, 2008.

[24] S. Kvatinsky *et al.*, "Mrl-memristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012, pp. 1–6.

[25] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 115–119, 2013.

[26] I. Vourkas *et al.*, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Transactions on Nanotechnology*, vol. 11, pp. 1151–1159, 2012.

[27] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 335–340.

[28] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.

[29] T. You *et al.*, "Exploiting memristive bifeo3 bilayer structures for compact sequential logics," *Advanced Functional Materials*, vol. 24, pp. 3357–3365, 2014.

[30] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[31] S. Kvatinsky *et al.*, "Magic–memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.

[32] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2016, pp. 1–6.

[33] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 2054–2066, 2014.

[34] R. Waser *et al.*, *Introduction to Nanoionic Elements for Information Technology*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016, pp. 1–30. [Online]. Available: http://dx.doi.org/10.1002/9783527680870.ch1

[35] E. Linn *et al.*, *ReRAM Cells in the Framework of Two-Terminal Devices*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016, pp. 31–48. [Online]. Available: http://dx.doi.org/10.1002/9783527680870.ch2

[36] D. Ielmini *et al.*, *Resistive switching: from fundamentals of nanoionic redox processes to memristive device applications*. John Wiley & Sons, 2015.

[37] R. Williams, "How we found the missing memristor," *IEEE Spectr.*, vol. 45, pp. 28–35, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1109/MSPEC.2008.4687366

[38] Y. Hayakawa *et al.*, "Highly reliable taox reram with centralized filament for 28-nm embedded application," in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, June 2015, pp. T14–T15.

[39] P. Junsangsri *et al.*, "A memristor-based TCAM (Ternary Content Addressable Memory) cell," in *2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Jul. 2014, pp. 1–6.

[40] A. Avizienis, "Signed-digit numbe representations for fast parallel arithmetic," *Electronic Computers, IRE Transactions on*, vol. EC-10, pp. 389–400, Sept 1961.

[41] B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *Computers, IEEE Transactions on*, vol. 37, pp. 1470–1476, 1988.

[42] D. Fey, "Using the multi-bit feature of memristors for register files in signed-digit arithmetic units," *Semiconductor Science and Technology*, vol. 29, p. 104008, 2014. [Online]. Available: http://stacks.iop.org/0268-1242/29/i=10/a=104008

[43] W. Kim *et al.*, "Multistate Memristive Tantalum Oxide Devices for Ternary Arithmetic," *Scientific Reports*, vol. 6, Dec. 2016. [Online]. Available: http://www.nature.com/articles/srep36652

[44] D. Fey *et al.*, "Using memristor technology for multi-value registers in signed-digit arithmetic circuits," in *Proceedings of the Second International Symposium on Memory Systems*, ser. MEMSYS '16. New York, NY, USA: ACM, 2016, pp. 442–454.

[45] D. Wust *et al.*, "Prototyping memristors in digital system with an fpga-based testing environment," in *to appear in Proc. IEEE/ 27th International Symposium on Power, and Timing, Modeling, Optimization and simulation (PATMOS)*, Sep. 2017, p. 7 pages.

[46] C. J. Xue, "Redesigning software and systems for nonvolatile processors on self-powered devices," in *Smart Sensors at the IoT Frontier*. Springer, 2017, pp. 107–123.

[47] F. Su *et al.*, "Design of nonvolatile processors and applications," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[48] L. Yavits *et al.*, "Resistive associative processor," *CAL*, 2015.

[49] H. Mostafa *et al.*, "Beyond spike-timing dependent plasticity in memristor crossbar arrays," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 926–929.

[50] F. Alibart *et al.*, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature communications*, vol. 4, p. 2072, 2013.

[51] A. Danowitz *et al.*, "Cpu db: recording microprocessor history," *Communications of the ACM*, vol. 55, pp. 55–63, 2012.

[52] A. Pedram *et al.*, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, pp. 39–50, 2017.

[53] H. A. Du Nguyen *et al.*, "On the implementation of computation-in-memory parallel adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[54] G. S. Snider, "Spike-timing-dependent learning in memristive nanodevices," in *2008 IEEE International Symposium on Nanoscale Architectures, NANOARCH 2008, Anaheim, CA, USA, June 12-13, 2008*, 2008, pp. 85–92. [Online]. Available: https://doi.org/10.1109/NANOARCH.2008.4585796

[55] T. Serrano-Gotarredona *et al.*, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," *Frontiers in Neuroscience*, vol. 7, p. 15, 2013. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2013.00002

[56] S. Kim *et al.*, "Nvm neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 17.1.1–17.1.4.

[57] C. K. K. Lim *et al.*, "Computing Image and Motion with 3-D Memristive Grids," in *Memristor Networks*, A. Adamatzky *et al.*, Eds. Springer International Publishing, 2014, pp. 553–558.

[58] Crossbar Inc., "Rethink data storage with 3d reram." [Online]. Available: https://www.crossbar-inc.com/en/

[59] Intel Corporation, "Revolutionizing memory and storage." [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html

[60] S. C. Bartling *et al.*, "An 8mhz 75μa/mhz zero-leakage non-volatile logic-based cortex-m0 mcu soc exhibiting 100vdd=0v with lt;400ns wakeup and sleep transitions," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Feb 2013, pp. 432–433.

[61] M. Barbareschi *et al.*, "Memristive devices: Technology, design automation and computing frontiers," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017 12th International Conference on*. IEEE, 2017, pp. 1–8.

[62] K. Ma *et al.*, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, pp. 32–40, 2015.

[63] B. Parhami, *Computer arithmetic*. Oxford university press, 1999, vol. 20, no. 00.

[64] M.-F. Chang *et al.*, "Endurance-aware circuit designs of nonvolatile logic and nonvolatile sram using resistive memory (memristor) device," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 2012, pp. 329–334.

[65] L. Xie *et al.*, "On the robustness of memristor based logic gates," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*. IEEE, 2017, pp. 158–163.

[66] Q. Xia *et al.*, "Memristor- cmos hybrid integrated circuits for reconfigurable logic," *Nano letters*, vol. 9, pp. 3640–3645, 2009.

3

PRIMITIVE LOGIC GATE

This chapter presents the design of primitive logic gates and studies the impact of device variability. First, it explores the complete logic gate space for Snider logic by parameterizing gates and identifying all possible combinations of such parameters. Subsequently, it proposes a novel logic design style, referred to as scouting logic. *This logic style performs logic operations by modifying standard memory read operations; they effectively reduce the device endurance requirement and improve the performance in terms of delay and power consumption. Finally, this chapter analyzes the robustness of primitive gates used in both Snider logic and scouting logic. The device variability, parasitic resistance and capacitance of nanowires, sneak path currents, and different memristor models are considered.*

The content of this chapter consists of the following research articles:

1. ***L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, Boolean Logic Gate Exploration for Memristor Crossbar, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Istanbul, Turkey, April, 2016, pp. 1-6*

2. ***L. Xie**, H.A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. Alfailakawi, S. Hamdioui, Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing,IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, July, 2017, pp. 151-156*

3. ***L. Xie**, H.A. Du Nguyen, J. Yu, M. Taouil, S. Hamdioui, On the Robustness of Memristor Based Logic Gates, IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Dresden, Germany, April, 2017, pp. 158-163*

3.1. INTRODUCTION

Primitive logic gates are the fundamental components for resistive computing. Many logic design styles have been proposed [15,22–36]. However, there are still some open questions: (i) What is the complete logic gate space for logic design based on memristive device? (ii) Can we develop any novel logic design styles? (iii) How can we guarantee the logic gates to behave correctly while the technology is not mature enough? This chapter focuses on these three questions.

Logic Gate Space Exploration: Snider logic [22] is a promising candidate due to its non-volatility, compatibility with the crossbar architecture, ability to do computation using only memristors, easy reuse of existing arithmetic designs, IP designs, and EDA flows. The author proposed three primitive operations: copy, inversion and NAND. However, these operations are only a subset of all the possible logic gates that can be implemented by Snider logic.

A Novel Logic Design Style: Many logic design styles use resistance to represent both input and output data, such as stateful implication logic [23], Snider logic [22], CRS logic [27], etc. However, in such logic design styles, a sequence of primitive operations is required to execute a simple logic gate (e.g., XOR) leading to the frequent device switching which reduces devices' endurance. Moreover, these logic designs suffer from a considerable delay hence low device switching speed.

Robustness Analysis: The logic design styles are facing some robustness issues due to the device variability, sneak path currents, parasitic resistance and capacitance of nanowires. Such robustness issues may cause the incorrect functionality of logic gates. Only limited work has been reported for robustness analysis. In [57], Kvatinsky et al. reported a set of constraints for individual implication logic gates. However, this work did not consider device variations, sneak path currents, and nanowire resistances. In [58], Zhu et al. reported a methodology to derive design constraints while considering sneak path currents within the crossbar. However, they did not consider dynamic switching process of memristors and other effects such as device variation and nanowire resistances. In addition, no such robustness analysis has been for Snider and scouting logic.

3.2. MAIN CONTRIBUTIONS

The main contributions in the above three aspects are as follows.

- *Complete Logic Gate Space Exploration* [52]: In order to explore the complete space of Snider logic gates, six parameters are extracted from the primitive gates; they are related to circuit structure, control voltage, fan-in, fan-out, device polarity, and data representation. By an exhaustive search of all the gate parameter combinations, the logic gate space is completely explored. As a result, several new gates are developed including AND, OR and NOR. In addition, we also find several new ways to implement the original gates used by Snider logic including NAND and copy operations. All such gates are verified with SPICE simulations.

- *A Novel Logic Design Style* [31]: Scouting logic performs logic operations by modifying standard memory read operations. Normally when a memory cell is read, a read voltage is applied to this cell to activate it. Subsequently, an input current for a sense amplifier is generated and it is compared with a reference current to distinguish between logic 1 and 0. As a result, the value stored in the memory cell is read out. Inspired by this read operation, scouting logic performs OR, AND and XOR operations, which are frequently used in bitwise logic operations. Instead of reading a single memory cell at a time, scouting logic activates the two inputs of the gate simultaneously. To perform different operations, we only need to use different current references. Therefore, we modified the standard sense amplifier to a reconfigurable one to support the required current reference. To illustrate the potential of scouting logic, the performance of scouting logic is compared with the state-of-the-art in terms of delay, power and area. The results show that scouting logic improves the delay and power consumption by at least a factor of 2.3 as compared to the state-of-the-art, while having similar or less area overhead.

- *Robustness Analysis* [31,53]: Robustness analysis of both Snider and Scouting logic gates. For Snider logic gates, a set of proper constraints are formulated to guarantee correct functionality of logic gates (e.g., AND). It is observed that the errors due to analytical parameter constraints are typically within 4.5% as compared to SPICE simulations. In addition, the impact of memristive device variability, parasitic resistance and capacitance of nanowires, sneak path currents, and different memristor models on its accuracy is investigated. When these realistic factors are within a certain range, the proposed constraints can still guarantee the correct functions of the logic gates. For Scouting logic gates, the impact of device variability is investigated. When the high and low resistance of the device vary within a certain range (e.g., around 10%), scouting logic can perform correctly. As the resistance variation range depends on the resistance values, a variation-resilient design methodology is proposed to select proper resistance values for a given failure rate. This proposed methodology can effectively enhance the robustness of the design to deal with resistance variation in scouting logic based design.

2016 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

Boolean Logic Gate Exploration for Memristor Crossbar

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering, Delft University of Technology, the Netherlands
Email: {L.Xie, H.A.DuNguyen, M.Taouil, S.Hamdioui, K.L.M.Bertels}@tudelft.nl

***Abstract*—Emerging technologies are under research as alternatives for next-generation VLSI circuits. One of the promising candidates is memristor due to its scalability, high integration density, non-volatility, etc. Different design styles of memristor-based logic circuits have been proposed. This paper first overviews these design styles and compares them using several criteria. Subsequently, it selects a promising candidate to explore its potential logic gate space. Thereafter, it derives control voltage constraints used to ensure correct logic gate functionality. The newly obtained logic gates are verified by SPICE simulations, and finally the performance of the memristor gates are compared with CMOS gates. The results show that the memristor gates, with reasonable technology improvements, are comparable to CMOS gates or even outperform them.**

I. INTRODUCTION

As CMOS technology gradually scales down to its intrinsically physical device limits, it faces major challenges [1] such as saturated performance enhancement, increased leakage power consumption, reduced reliability, etc. To address such challenges, new technologies (e.g., memristors, nanotube, graphene transistors, etc. [2]) have been proposed as alternatives for next-generation VLSI circuits. Among these technologies, the memristor is one of the promising candidates. Massive memristors can be mapped on a crossbar architecture, where memristors are located between horizontal and vertical nanowire junctions [3]. The memristor crossbar is able to provide great scalability, high integration density, non-volatility, etc [3]. Several potential applications have been proposed including non-volatile memory [2,3], neuromorphic circuits [2,3] and novel computing paradigms for data-intensive applications (e.g., computation-in-memory [4,5]). To realize novel computing paradigms, it is pivotal to design fundamental logic circuits/gates.

Four types of memristor-based logic circuits have been proposed: (i) threshold logic [6], (ii) majority logic [6], (iii) material implication logic [7], and (iv) Boolean logic [8]; each logic type consist of different design styles. Among them, Snider Boolean logic (SBL) [9] is a promising candidate for memristor crossbar as shown later in this paper. The author of SBL proposed three primitive operations: copy, inversion and NAND. However, these operations are only a subset of all the possible logic gates that can be implemented by SBL. In addition, the author did not notice that these logic gates only function correctly when the applied control voltages satisfy certain constraints.

This paper explores all the possible logic gates for SBL. Thereafter, it derives the control voltage constraints by formulating the switching conditions of memristors during operation. The main contributions of this paper consist of:

- A brief overview of existing memristor-based logic circuits.
- The logic gate space exploration for SBL.
- The derivation of the control voltage constraints to ensure correct functionality of logic gates.
- A comparison between memristor and CMOS logic gates.

The remainder of this paper is organized as follows. Section II overviews memristor logic circuits and selects SBL as a promising candidate. Section III explores the SBL logic gate space. Section IV derives the control voltage constraints. Section V verifies new memristor gates and compares memristor and CMOS gates. Finally, Section VI concludes the paper.

II. OVERVIEW OF MEMRISTOR LOGIC CIRCUITS

This section first overviews the-state-of-art of memristor logic circuits. Thereafter, it selects SBL as a promising candidate to implement memristor crossbar logic circuits.

The memristor logic circuits are briefly overviewed by considering criteria related to *technology and design*.

Two **technology** criteria are considered: crossbar compatibility (Xbar) and computing technology requirement (Cmp.Tech.). Several memristor logic circuits have been proposed in particular for the crossbar architecture and therefore, they are compatible with crossbar (Y); in contrast, others may not be compatible (N). The memristor logic circuits consist of a computing (e.g., logic gates) part, and a CMOS control part (e.g., control logic, clock, etc.). Some logic circuits require only memristors (M) for the computing part, while others require both CMOS and memristors (CM). This is captured by computing technology requirement.

The considered **design** criteria are:

Logic Type: The logic type specifies the kinds of operations that are performed; e.g., Boolean logic includes operations such as AND, OR, etc. Five logic types have been implemented with memristors; they are Boolean logic (Bool), implication logic (IMP), threshold logic (TH), majority logic (MAJ) and hybrid logic (Hyb.). In hybrid logic, different logic types are merged (e.g., Boolean and implication logic are combined in [18]).

978-1-5090-0336-5/16/$31.00 ©2016 IEEE

TABLE I: Features of Memristor-Based Logic Circuits

Ref	Name	Abbr.	Technology		Design						
			Xbar	Cmp. Tech.	Logic Type	Logic Gate	Data Sig.	Usage	Syn.	No. Step	NV
[8]	Hybrid Transistor/Memristor Boolean Logic	HTMBL	N	CM	Bool	NAND	V	CFG	AS	S	N
[10]	Memristor Ratioed Logic	MRL	N	CM	Bool	AND,NAND,OR,NOR	V	CMP	AS	S	N
[11]	mLogic	mLogic	N	M	Bool	AND,NAND,OR,NOR,NOT	C	CMP	CLK	M	Y
[12]	Domain Wall Logic	DWL	N	M	Bool	AND,OR,NOT	C	CMP	CLK	M	Y
[6]	Memristive Programmable Logic Array	MPLA	Y	CM	Bool	AND,OR,NOT	V	CFG	AS	S	N
[13]	CMOS-like Configurable Memristor Logic	CCML	Y	M	Bool	AND,NAND,OR,NOR,NOT	V	CMP	AS	S	Y
[14]	Stateful Logic Pipeline Architecture	SLPA	Y	M	Bool	OR,NOR	R	CMP	CTR	M	Y
[9]	Snider Boolean Logic	SBL	Y	M	Bool	CPY,NOT,NAND	R	CMP	CTR	M	Y
[15]	BRS/CRS Crossbar Logic	BCCL	Y	M	IMP	CIMP,NIMP	V	CMP	CTR	M	Y
[7]	Material Implication Logic	MIL	Y	M	IMP	IMP	R	CMP	CLK	M	Y
[6]	Charge Sharing Threshold Gate	CSTG	N	CM	TH	TG	V	MEM	AS	S	N
[6]	Current Mirror Threshold Gate	CMTG	N	CM	TH	TG	V	MEM	AS	S	N
[16]	Programmable CMOS/Memristor Threshold Logic	PCMTL	N	CM	TH	TG	V	MEM	CLK	S	N
[6]	Memristive Programmable Threshold Logic Array	MPTLA	Y	CM	TH	TG	V	MEM	CLK	S	N
[17]	All Spin Logic	ASL	N	M	MAJ	MAJ,CPY,NOT	C	CMP	CLK	M	Y
[6]	Memristive Programmable Majority Logic Array	MPMLA	Y	CM	MAJ	MAJ	V	MEM	CLK	S	N
[18]	Memristive Stateful Logic	MSL	Y	M	Hyb.	IMP,CNIMP,AND,OR	R	CMP	CTR	M	Y

Logic Gate: Logic gates compute primitive operations used to build complex digital functions. For Boolean logic, the basic operations are AND, OR, NAND, NOR, and NOT. Note that not all the Boolean logic circuits can implement all gates. For implication logic, implication (IMP, $f{=}\overline{A}{+}B$ where A and B are inputs, and f the output), converse implication (CIMP, $f{=}A{+}\overline{B}$), nonimplication (NIMP, $f{=}A{\cdot}\overline{B}$), and converse nonimplication (CNIMP, $f{=}\overline{A}{\cdot}B$) gates have been proposed [19]. Threshold logic uses threshold gates (TG) and majority logic majority gates (MG).

$$f = \begin{cases} 1 & \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

$$f = \begin{cases} 1 & \sum_{i=1}^{n} x_i > (n+1)/2 \\ 0 & \text{Otherwise} \end{cases} \quad (2)$$

Eq.1 and Eq.2 describe a TG and MG, respectively, where n presents the number of inputs, x_i the input, w_i the weight of input x_i, and T the threshold. Finally, some logic circuits may use a copy (CPY) operation to transmit data within the crossbar.

Data Signal (Data Sig.): The data in memristor circuits is presented either by a voltage (V) (i.e., different voltage levels), a current (C) (i.e., different current directions or values) or a resistance (R) (i.e., high or low resistance).

Memristor Usage (Usage): A memristor can be used as a digital computing switch (CMP), a resistive memory (MEM) or a configuration switch for signal routing in programmable logic array (CFG).

Synchronization (Syn.): Some logic circuits use a clock (CLK) or CMOS controller (CTR) for synchronization, while others operate asynchronously (AS).

Number of Steps: Logic circuits may complete the operation in a single (S) step or multiple steps (M).

Non-volatility (NV): The non-volatility indicates whether a logic circuit can store its results when it is powered down.

Table I summarizes the features of each design style. For instance, MRL uses hybrid technology to implement Boolean logic; it uses individual memristors based on a voltage; its basic gates are AND, NAND, OR and NOR gates where each memristor is used as a computing switch; MRL works asynchronously, computes the result within one cycle without having non-volatility. Note that Table I reflects relationships between different features; e.g., most hybrid CMOS/memristor circuits calculate the results within one cycle, while memristor-only circuits require several cycles.

SBL is one of the most promising candidates for memristor crossbar due to the following reasons. Technology-wise, SBL is compatible with memristor crossbar which provides great scalability and high integration density. Next, SBL's computing part is implemented only by memristor (no CMOS logic required) which potentially enables novel computing paradigms [4,15]. Design-wise, SBL is based on Boolean logic; this is preferred as it enables the resue of existing arithmetic designs, IP designs, and EDA flows. In addition, SBL is non-volatile, and therefore, has the potential to support lower-power designs as it can be powered-off when it is not used. Note that other criteria that are not used in this work (e.g., latency) may lead to different outcomes.

III. LOGIC GATE SPACE OF SNIDER BOOLEAN LOGIC

This section first describes the memristor model used by SBL. Thereafter, it presents the existing SBL gates. By investigating these gates, gate parameters are extracted which are subsequently used to explore the potential logic gate space.

A. Memristor Model

The current-voltage relation of the ideal memristor used in SBL [9] is shown in Fig. 1(a). In case the absolute value of the voltage across the device is greater than its threshold voltage V_{th}, the memristor switches from one resistive state to another. Otherwise, it stays in its current resistive state. Typically, a memristor requires two different V_{th} values to switch from high (R_H) to low (R_L) resistance (SET), and low to high resistance (RESET) [20]; see Fig. 1(b) and (c). The black squares at the top edge of the memristors present the positive terminal. For simplicity, we assume that this model has the same absolute V_{th} value for both SET and RESET.

B. Existing SBL Gates

The authors in SBL [9] proposed three different logic gates: copy (CPY), inversion (INV) and NAND as shown in Fig. 2.

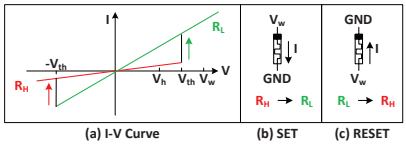


Fig. 1: Memristor model [9].

Each gate consists of one or multiple input memristors and an output memristor; the output memristor (surrounded by a dashed-line box in Fig. 2) is initialized to R_H prior to execution. For brevity, this initialization is not shown. The nanowire that connects the input and output memristors is floating; V_x presents the voltage of this floating nanowire and V_{om} the voltage across the output memristor. Both V_x and V_{om} are used to explain the working principle. In addition, INV and NAND gates requires an extra resistor R_s attached to the floating nanowire as shown in Fig. 2(c) and (e), respectively; R_s must satisfy $R_L \ll R_s \ll R_H$ to guarantee that V_x is close to the desired voltage for correct operation [9].

SBL logic gates use a high (R_H) and low (R_L) resistance to represent logic 1 and 0. To control these gates, three different voltages are required: V_w, V_h, and *GND* (G); see Fig. 1(a). V_w is used to program the resistance of a memristor; V_h is used to control primitive operations (e.g., NAND [9]), or used as half-select voltages to reduce the impact of sneak path currents [21]. The typical relationship between V_w, V_h, G and V_{th} is $0<V_h=\frac{V_w}{2}<V_{th}<V_w<2V_{th}$. This is the minimum requirement to ensure the functionality of logic gates.

CPY is given as an example as other logic gates work in a similar way. Voltages G and V_w are applied to the input and output memristors, respectively. In case the input is 1 (R_H), $V_x=V_w/2$, and hence, $V_{om}=V_w-V_x=V_w/2<V_{th}$ (see Fig. 2(a)). As a result, the output memristor stays at 1. In case the input is 0 (R_L), $V_x\approx 0$ and $V_{om}\approx V_w-V_x\approx V_w>V_{th}$ (see Fig. 2(b)). As a result, the output memristor switches to 0.

C. Gate Parameters

To explore the logic gate space of SBL, several gate parameters are extracted from the SBL gates. These parameters are desribed next.

Structure: From the CPY and INV gates of Fig. 2(a) and (c), it is observed that the gates have a different structure due to R_s; R_s can be regarded as an extra terminal of the schematic. Hence, the schematic without R_s (see for example CPY in Fig. 2(a)) is referred to as *2 terminal (2T)*, while the schematic with R_s is referred to as *3 terminal (3T)* (e.g., INV gate in Fig. 2(c)).

Control Voltage: From the CPY and INV gates of Fig. 2(a) and (c), it is observed that the gates are controlled with different voltages; for instance, the *GND* and V_h are applied to input memristors of CPY and INV, respectively. The control voltages impact the functionality of the logic gates. For instance, Fig. 2(g) and (h) swap the input and output control voltages with respect to Fig. 2(a). As a result, this gate is not able to copy data from the input to output

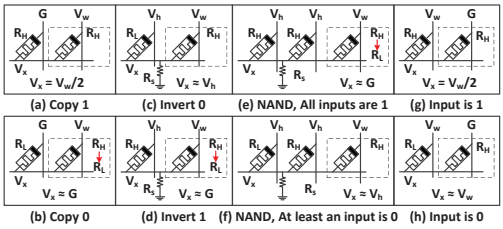


Fig. 2: Basic Operations

memristors. In general, we denote the control voltages using the tuple (V_{ci},V_{co}), where V_{ci} and V_{co} present the voltage applied to the input and output memristors, respectively; e.g., the control voltages of CPY in Fig. 2(a) is denoted by (G,V_w).

Fan-in (FI): From the INV and NAND gates of Fig. 2(c) and (e), it is observed that the fan-in (the number of inputs) leads to different functions; INV has a single fan-in (*SFI*), while NAND a multi fan-in (*MFI*). The number of inputs is denoted by n_i.

Fan-out (FO): All the existing SBL logic gates have only one output memristor; these gates are referred to as single fan-out (SFO) gates. Here, we extend the SFO gates to multi fan-out (MFO) gates by applying the same control voltages to multiple output memristors. The number of outputs is denoted by n_o.

Polarity: All input and output memristors of the existing SBL gates have their negative terminal connected to the floating nanowire, which is referred to as *negative polarity* (see also Fig. 2). Here, we also explore the case where all the positive termnals of the input and output memristors are connected to the floating nanowire; this is referred to as *positive polarity*. We limit the exploration to the case where either all positive or all negative terminals of both input and output memristors are connected to the vertical nanowires for manufacturing reasons [21].

Data Representation: All existing SBL gates use R_H and R_L to represent a logic 1 and 0, which are referred to as an *RH1* data representation. Here, we will also explore the case when R_H is used to represent logic 0 and R_L logic 1, which are referred to as *RH0*.

Based on the above gate parameters, we propose generic gate schematics for SBL as shown in Fig. 3(a) for 2T and (b) for 3T. In the figure, $M_{i,k}$ ($1<k<n_i$) and $M_{o,p}$ ($1<p<n_o$) present the resistances of the input and output memristors, respectively. These two schematics are simplified by their equivalent circuits obtained through Kirchhoff's current law; note that all the input/output memristors are driven by the same control voltages V_{ci}/V_{co}. The results are depicted in Fig. 3(c)-(d) where $M_{i,eq}$ and $M_{o,eq}$ present the equivalent resistance of input and output memristors, respectively. The equivalent circuits will be used to derive control voltage constraints in section IV.

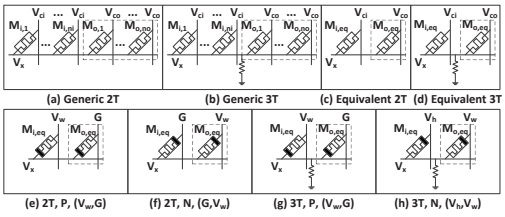


Fig. 3: 2T and 3T Logic Gates.

D. Logic Gate Space

The logic gate space is explored through an exhaustive search of all the gate parameters. The SFI/SFO gate space is first explored, and subsequently extended for MFI/MFO gates.

The SFI/SFO logic gate space is explored by using the following gate parameters: structure, polarity and control voltages; data representation is not considered (e.g., is set to RH1) as SFI/SFO logic gates can only either be CPY or INV gates; therefore, the data representation (RH1 or RH0) has no impact on them. This leads to total 24 (2 structures x 2 polarities x 3! control voltages) combinations. Among all the combinations, only four combinations are useful as shown in Fig. 3 (e)-(h); the first three gates in Fig. 3 (e)-(g) are CPY gates, while the last one in Fig. 3 (h) is an INV gate. Note that although no new functions are formed, our approach introduces two novel CPY gate implementations as shown in Fig. 3 (e) and (g). All the other combinations of the gate parameters ended up in useless gates, such as a gate shown in Fig. 2(g)-(h) where the output is always 0.

The MFI/MFO gate space is derived from the SFI/SFO gate space by extending it with the remaining gate parameters (i.e., fan-in, fan-out and data prepresentation) that are not considered in the SFI/SFO gate space exploration. Note that the fan-out does not impact the gate operations, whereas fan-in and data representation may do so. For instance, by *only* changing the SFI of the INV gate of Fig. 2(c) to a MFI gate the NAND gate of Fig. 2(e) is obtained. This can be explained by the parallel input memristors and their equivalent resistance $M_{i,eq}$; it can be either a relatively high resistance ($=\frac{R_H}{n_i}$) when all inputs are 1, or a relatively low resistance ($\approx\frac{R_L}{n_{iL}}$) when n_{iL} inputs are 0. From this, we observe that the parallel input memristors behave like an AND gate. Therefore, an AND gate can be realized by replacing the single fan-in of CPY in Fig. 4(b) with a multi fan-in. Similarly, in the case of RH0, the multi fan-in introduces OR and NOR gates. Note that the OR and NOR gates are the same as the AND and NOR gates except that they differ in data representation.

Fig. 5 shows the explored gate space with their gate parameters. These gates support basic computation and communication. In the figure, the original SBL gates proposed in [9] are highlighted by boxes. Compared to the original SBL, our approach (i) identifies new gates (such as AND, OR, NOR gates), (ii) shows different ways to design them (using 2T or 3T structure and different polarities), and (iii) introduces multi fan-out gates. Overall, these new gates have the potential to design logic circuits more efficiently than the original SBL gates.

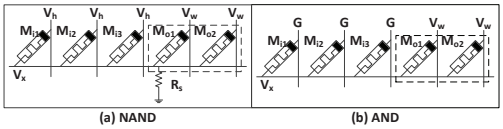


Fig. 4: Multi Fan-out Logic Gates.

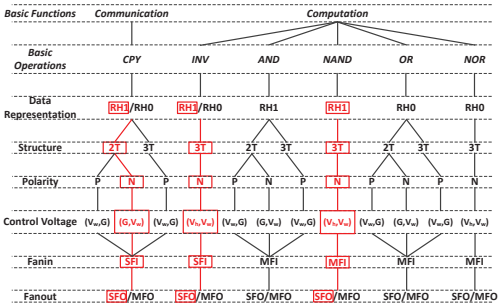


Fig. 5: Logic Gate Space.

IV. CONSTRAINTS OF CONTROL VOLTAGES

This section formulates the control voltage constraints of all the logic gates of Fig. 5. The logic gates only function correctly when their control voltages are in certain range. The voltage constraints are derived from the switching conditions of both input and output memristors. We assume that each memristor behaves as a resistor at each moment in time and ignore the resistance and capacitance of the nanowires.

The functionality of the gate is guaranteed when the switching conditions of the input and output memristors are satisfied. For all the logic gates, the input memristors should stay in their current resistive state independent of the gate function. The output memristors, which are initialized to R_H prior to evaluation, either stay at R_H or switch to R_L depending on the input values and gate type.

To illustrate the derivation of constraints, as an example we discuss the AND gate with the following gate parameters: RH1, 2T, P, MFI and MFO (see also Fig. 5). Fig.3(e) shows the equivalent circuit of this AND gate. In case all the inputs are 1 (R_H), the outputs of the AND gate must remain 1 (R_H) and $M_{i,eq}=\frac{R_H}{n_i}$ and $M_{o,eq}\approx\frac{R_H}{n_o}$. To ensure that all the input and output memristors stay logic 1 (R_H), the voltage across the input (V_{im}) and output (V_{om}) memristors should be less than V_{th} as shown in Eq.3.

$$V_{im} = V_x - V_{ci} = V_x - V_w < V_{th} \qquad (3)$$
$$V_{om} = V_x - V_{co} = V_x - 0 < V_{th}$$

In Eq.3, only the voltage of the floating nanowire V_x is unknown. To express V_x, Kirchhoff's current law is applied to

TABLE II: Control Voltage Constraints

2T Structure, Positive/Negative Polarity
$1+\frac{1}{r_R}\frac{n_o}{n_{iL}} < r_{wt} < min\{1+\frac{n_o}{n_i}, 1+\frac{n_{iL}}{n_o}\}$
3T Structure, Positive Polarity
$1+\frac{1}{r_R}\frac{n_o}{n_{iL}}+\frac{r_{Ls}}{n_{iL}} < r_{wt} < min\{1+\frac{n_o}{n_i}+\frac{r_{Hs}}{n_i}, 1+\frac{n_{iL}}{n_o+r_{Ls}}\}$
3T Structure, Negative Polarity
$1+\frac{n_i r_{ht}+n_o}{n_i+r_{Hs}} < r_{wt} < min\{1+\frac{r_R n_{iL} r_{ht}+n_o}{r_R n_{iL}+r_{Hs}}, 1+\frac{n_i}{n_o}+\frac{r_{Ls}}{n_o}+(1+\frac{r_{Ls}}{n_o})r_{ht}\}$

the circuit of Fig. 3(e), and solved for V_x as shown in Eq.4.

$$\frac{V_w - V_x}{M_{i,eq}} = \frac{V_x - 0}{M_{o,eq}} \Rightarrow V_x = \frac{M_{o,eq}V_w}{M_{o,eq}+M_{i,eq}} \quad (4)$$

Therefore, V_{im} and V_{om} can be rewritten as:

$$\Rightarrow V_{im} = -\frac{M_{i,eq}V_w}{M_{i,eq}+M_{o,eq}}; V_{om} = \frac{M_{o,eq}V_w}{M_{o,eq}+M_{i,eq}} \quad (5)$$

According to Eq.5, the condition $V_{im}<V_{th}$, which specifies that the input memristors must stay R_H, is always true as $V_{im}<0<V_{th}$. The condition $V_{om}<V_{th}$ for the output memristors is rewritten in Eq.6 by substituting $M_{i,eq}=\frac{R_H}{n_i}$ and $M_{o,eq}=\frac{R_H}{n_o}$, respectively.

$$V_{om} = \frac{n_i}{n_i+n_o}V_w < V_{th} \quad (6)$$

In case at least one input of the AND gate is 0 (R_L), four conditions must be satisfied. All the inputs must stay at their current resistive states before and after the outputs switch, and the outputs must switch from logic 1 to 0 and stay at logic 0 after switching. Before the outputs switch, $M_{i,eq}\approx\frac{R_L}{n_{iL}}$, $M_{o,eq}=\frac{R_H}{n_o}$ and V_x is obtained by substituting these equivalent resistances in Eq.4. To ensure that the input memristors stay at their current resistive states, $-V_{th}<V_{im}<V_{th}$ (see Fig. 1(a)); to switch the output memristors to R_L, $V_{om}>V_{th}$. The relevant constraints are expressed in Eq.7 where $r_R=\frac{R_H}{R_L}$.

$$-V_{th} < V_{im} = -\frac{V_w}{1+r_R\frac{n_{iL}}{n_o}} < V_{th}, V_{om} = \frac{V_w}{1+r_R\frac{n_o}{n_{iL}}} > V_{th} \quad (7)$$

After the output memristors switch to R_L, the $M_{i,eq}$ remains the same ($M_{i,eq}\approx\frac{R_L}{n_{iL}}$), while $M_{o,eq}$ changes to $\frac{R_L}{n_o}$. To ensure that the input memristors stay at their current resistive states, $-V_{th}<V_{im}<V_{th}$ (see Fig. 1(a)); to prevent the output memristors from switching back to R_H, $V_{om}>-V_{th}$. Note that $V_{om}>-V_{th}$ is always true as $V_{om}>0$ (see Eq.5).

$$-V_{th} < V_{im} = -\frac{V_w}{1+\frac{M_{o,eq}}{M_{i,eq}}} = -\frac{V_w}{1+\frac{n_{iL}}{n_o}} < V_{th} \quad (8)$$

All above constraints are further reduced to Eq.9 where $r_{wt}=\frac{V_w}{V_{th}}$ and $r_{ht}=\frac{V_h}{V_{th}}$.

$$1+\frac{1}{r_R}\frac{n_o}{n_{iL}} < r_{wt} < min\{1+\frac{n_o}{n_i}, 1+\frac{n_{iL}}{n_o}\} \quad (9)$$

Note that $n_{iL}=1$ is the worst case condition. The constraints for other gates can be derived in a similar way. All the relevant constraints of each gate are summarized in Table II where $r_{Hs}=\frac{R_H}{R_s}$ and $r_{Ls}=\frac{R_L}{R_s}$.

TABLE III: Memristor Technology and Simulation Parameters

Technology						
F (nm)	D (nm)	R_L (Ω)	R_H (Ω)	V_{th} (V)	Area (F^2)	T_{sw} (ps)
50	8 (5)	200k (4M)	400M	1.5	4	527 (10.2)
Simulation						
V_w (V)		V_h (V)		R_s (Ω)		
1.95		0.975		2M (40M)		

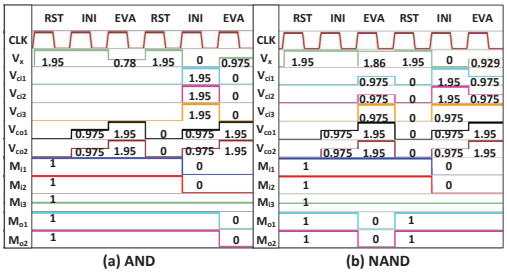


Fig. 6: Simulation Results of Logic Gates.

V. SIMULATION RESULTS AND EVALUATION

This section first verifies the proposed logic gates and their constraints using SPICE simulations; subsequently it discusses the potentials of SBL gates by comparing them with CMOS gates in terms of area, delay and power/energy consumptions.

A. Simulation

The simulation model of the SBL gates contains a CMOS controller, voltage drivers, and a memristor crossbar. The controller is used to determine the control voltage that must be applied to each nanowire; it is based on a state machine consisting of four states: IDLE, RST (reset all the memristors to R_H), INI (initialize the input memristors to R_H or R_L) and EVA (evaluate the logic gates). The voltage drivers are implemented with three pass transistors and three voltage sources [21]. The memristor model (see Section III.A), controller and voltage drivers are described by Verilog-A modules, while the memristor crossbar by a SPICE netlist.

Table III summarizes both the memristor technology and simulation parameters; the technology parameters R_L, R_H, V_{th}, feature size (F), area (A) and thickness of TaO_x film (D) are extracted from the physical devices in [22], while the switching time (T_{sw}) is derived from the method used in [23]; the values of the technology parameters in brackets can be ignored for now and are explained later. The simulation parameters consist of the control voltages and R_s, which are determined by the proposed constraints in section IV.

To verify the proposed logic gates, the AND and NAND gates of Fig. 4 with three inputs and two outputs are used as examples. Fig. 6 shows the simulation results. In the waveform, the voltages $V_{ci\#}$, $V_{co\#}$, and V_x are shown in addition to the input and output values; R_H and R_L present logic 1 and 0.

The accuracy and effectiveness of the voltage constraints are mainly determined by the formulation of V_x. Therefore, the formulas in Section IV are used to calculate V_x, and the

TABLE IV: V_x at State EVA

Cases	AND		Cases	NAND	
	Cal. (mV)	SPICE (mV)		Cal. (mV)	SPICE (mV)
All 1	780	780	Not All 1	929.1	929
Not All 1, B	0.974	0.974	All 1 B	33.3	33.3
Not All 1, A	974.9	975	All 1, A	1856.5	1860

calculated results are compared with the simulation results as shown in Table IV. The table shows that for the AND and NAND gates the calcuated (cal.) and simulated (SPICE) values of V_x both before (B) and after switching (A) have an error of only a couple of mVs. The simulation results show that the functionality of gates perform correctly, and the constraints are effective to determine the values of control voltages.

B. Comparison with CMOS gates

To investigate the potentials of the SBL gates, the INV and 2-input NAND gates are compared to 32nm FinFET gates [3,24] in terms of area, delay, power/energy consumption during a single operation. The area of each SBL gate includes the sum of both the transistors (T) required for the control and memristors (M) used for computation; the voltage drivers are implemented using the same technology as CMOS gates; these voltage drivers can be shared by several (e.g., 5) SBL gates as they can use the same nanowires in the crossbar; the area of R_s (R) is not considered. In addition, we only consider the evaluation step (e.g., EVA in Fig. 6) and therefore, the controller is ignored. The delay of each SBL gate is $3{\cdot}T_{sw}$ as it requires three states for computation (see Fig. 4). The power consumption of the SBL gates is the sum of the power consumption of each memristor and R_s (which are calculated by $P = V^2/R$ where V is the voltage across the memristor, R the memristor's resistance at that moment; V is calculated by the voltage formulas in Section IV). Table V shows the evaluation results for both SBL and FinFET gates. For SBL gates, we consider two cases based on (a) existing and (b) future memristor technology. The future memristor technology parameters can be found by the numbers in the brackets of Table III. Comparing to the same type of FinFET gates, SBL gates perform worse in terms of delay and energy consumption, and may consume more power for exisiting memristor technology. For example, the SBL inverter consumes less power (652.9 nW) than the CMOS inverter (1067 nW) with the almost same area. However, the SBL inverter has longer (about 150x) delay than the CMOS inverter, and this leads to a higher energy consumption. In contrast, SBL gates with future technology parameters (e.g., with higher R_L and less thickness of TaO_x) can outperform FinFET gates in all metrics. For instance, the delay of the SBL inverter using future memristor technology is reduced to around 31ps, which is comparable to the CMOS inverter; therefore, the SBL inverter consumes around 28x less energy than CMOS. Overall, the SBL gates are potential to implement logic gates consume less area and power/energy with a slight delay panelty with reasonable technology improvements.

TABLE V: 32nm CMOS and Memristor Comparison

Inputs	Tech.	Gates	No. of Devices			Area (μm^2)	Delay (ps)	Power (nW)	Energy (fJ)
			T	M	R				
1	INV	CMOS	2	0	0	0.0794	10.6	1067	0.0113
		Existing	6*	2	1	0.0795	1581	652.9	0.3439
		Future					30.9	36.73	0.0004
2	NAND	CMOS	4	0	0	0.1590	14.9	194	0.0029
		Existing	9*	3	1	0.1014	1581	549.5	0.2895
		Future					30.9	31.13	0.0003

*These transistors can be shared among 5 gates.

VI. CONCLUSION

This paper first overviewed the existing memristor logic circuits. Thereafter, it explored the logic gate space for Snider Boolean logic, which is one of the promising candidates for logic circuit design style for memristor crossbars. Compared to CMOS gates, memristor-based gates show potentials to achieve competitive performance with reasonable technology improvements. The Snider Boolean logic gates can be used for both computation and communication; therefore, they are a promising alternative for future VLSI circuits.

REFERENCES

[1] S. Borkar, "Design perspectives on 22nm cmos and beyond," in *DAC*. ACM, 2009, pp. 93–94.

[2] W. Zhao *et al.*, "Nanodevice-based novel computing paradigms and the neuromorphic approach," in *ISCAS*. IEEE, 2012, pp. 2509–2512.

[3] ITRS report. [Online]. Available: http://www.itrs.net

[4] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015, p. 199.

[5] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *DATE*, 2016, p. 188.

[6] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.

[7] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.

[8] J. Borghetti *et al.*, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *PNAS*, vol. 106, pp. 1699–1703, 2009.

[9] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.

[10] S. Kvatinsky *et al.*, "MRL:memristor ratioed logic," in *CNNA*. IEEE, 2012, pp. 1–6.

[11] D. Morris *et al.*, "mlogic: Ultra-low voltage non-volatile logic circuits using STT-MTJ devices," in *DAC*. ACM, 2012, pp. 486–491.

[12] J. A. Currivan *et al.*, "Low energy magnetic domain wall logic in short, narrow, ferromagnetic wires," *Magnetics Letters, IEEE*, vol. 3, pp. 3 000 104–3 000 104, 2012.

[13] I. Vourkas *et al.*, "A novel design and modeling paradigm for memristor-based crossbar circuits," *Nanotechnology, IEEE Transactions on*, vol. 11, pp. 1151–1159, 2012.

[14] K. Kim *et al.*, "Stateful logic pipeline architecture," in *ISCAS*. IEEE, 2011, pp. 2497–2500.

[15] E. Linn *et al.*, "Beyond von Neumann: logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, p. 305205, 2012.

[16] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 115–119, 2013.

[17] B. Behin-Aein *et al.*, "Proposal for an all-spin logic device with built-in memory," *Nature nanotechnology*, 2010.

[18] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[19] A. N. Whitehead *et al.*, *Principia mathematica*. University Press, 1912, vol. 2.

[20] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 211–221, 2013.

[21] K.-H. Kim *et al.*, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, pp. 389–395, 2011.

[22] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Advanced Materials*, vol. 23, pp. 5633–5640, 2011.

[23] M. Zangeneh *et al.*, "Performance and energy models for memristor-based 1T1R RRAM cell," in *GLVLSI*. ACM, 2012, pp. 9–14.

[24] C. Meinhardt *et al.*, "FinFET basic cells evaluation for regular layouts," in *LASCAS*. IEEE, 2013, pp. 1–4.

Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing

Lei Xie, H.A. Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi*, Said Hamdioui
Laboratory of Computer Engineering, Delft University of Technology, the Netherlands
*Computer Engineering Department, Kuwait University, Kuwait
Email: {L.Xie,H.A.DuNguyen,J.Yu-1,M.Taouil,S.Hamdioui}@tudelft.nl; alfailakawi.m@ku.edu.kw

3

***Abstract*—Memristor technology is a promising alternative to CMOS due to its high integration density, near-zero standby power, and ability to implement novel resistive computing. One of the major limitations of these architectures is the limited endurance of memristor devices, especially when a logic gate requires multiple steps/switching to execute the logic operations. To alleviate the endurance requirement and improve the performance, we present a novel logic design style, called *scouting logic* that executes any logic gate by only reading the memristor devices and without changing their states. Hence, no impact on the memristors' endurance. The proposed design is implemented using two styles (current and voltage based). To illustrate the performance of scouting logic based designs, the area, delay, and power consumption are analyzed and compared with state-of-the-art. The results show that scouting logic improves the delay and power consumption by at least a factor of 2.3, while having similar or less area overhead. Finally, we discuss the potential applications and challenges of scouting logic.**

I. INTRODUCTION

As CMOS technology is being continuously scaled down towards its physical limits, it suffers from major challenges such as saturated performance improvement, increasing leakage power, etc [1,2]. Emerging technologies, such as memristors, nanotube, graphene transistors [1], are under research as an alternative to CMOS technology. Memristor is one of the most promising candidates due to its great scalability, high integration density, and its near-zero standby power [1,3,4]. Novel memristor-based computing architectures [5–7] have been proposed as an alternative to today's von-Neumann architectures for big-data applications. Preliminary results of these resistive architectures show several orders of magnitude improvement for different metrics such as energy and area efficiency [5,6]. Big-data applications typically need to process large volume of data resulting in frequent device switching which poses as a concern for the endurance-limited memristor technology [3]. As a result, a logic design style with less switching frequency is required.

Recently, three types of memristor-based logic have been proposed; they can be classified into [4]: threshold/majority [8,9], implication [10,11] and Boolean logic [7,12,13]. Since threshold and majority logic use voltages to represent data, they are more applicable to von-Neumann architectures [8,9]. Due to the fact that implication and Boolean logic designs represent data as resistances, it is more efficient to use such approaches in resistive computing architectures [12,13]. However, in such logic designs, a sequence of primitive operations is required to execute a simple logic gate (e.g., XOR) leading to frequent device switching which reduces devices' endurance. Moreover, these logic designs suffer from a considerable delay hence low speed. To operate logic gates with limited endurance, the authors of [7] proposed an approach to implement logic operations using read operations by modifying the sense amplifier. Unlike previous approaches in [7], AND and OR gates are executed in one read operation while the XOR gates executed in two. However, their approach is area-inefficient and requires two steps to execute in the worst case.

In this work, we propose a novel logic design style, called scouting logic to enhance the performance of memristor-based logic circuits by limiting all gate executions to a single read operation. The current through the equivalent gate input resistance or the voltage drop over it is compared with a reference signal. The proposed design style is implemented using two types of sense amplifiers (current and voltage based) and their performance in terms of area, delay and power consumption are investigated. Moreover, designing robust scouting logic in presence of variations is discussed.

The remainder of this paper is organized as follows. Section II provides a background on memristors and briefly describes the state-of-the-art in memristor-based logic designs. Section III presents the working principle of scouting logic. Section IV verifies the designs of scouting logic using simulations and evaluates their performance. Section V discusses the potential applications and challenges of scouting logic. Finally, Section VI concludes the paper.

II. BACKGROUND

A. Memristor

Fig. 1 (a) shows the typical current-voltage relation of a bipolar memristor [3]. A memristor switches from one resistive state to another (i.e., a write operation) when the absolute value of the applied voltage (V_w) across the device is greater than its threshold voltage (see Fig. 1(b)-(c)). Otherwise, it stays in its current resistive state. Normally, a memristor requires different switching threshold voltages for SET (V_{ts}) and RESET (V_{tr}) operations where $V_{tr}<V_{ts}$ [3]. Reading the

978-1-5090-6762-6/17/$31.00

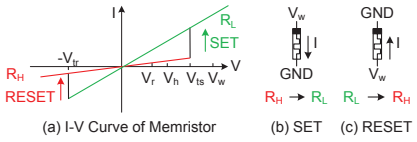


Fig. 1: Memristive Behaviour.

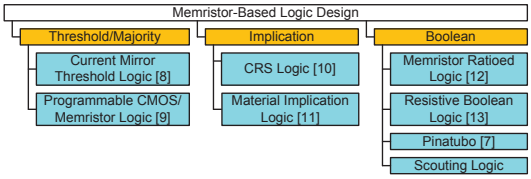


Fig. 2: Classification of Memristor-Based Logic

memristor is performed by applying the read voltage V_r; based on the resistance a low or high current will flow through the device. An additional control voltage V_h is required in some of the Boolean and implication logic operations [11,13]. Typically, the control voltages should satisfy the relation: $0<V_r<V_h=V_w/2<V_{tr}<V_{ts}<V_w<2V_{tr}$ (see Fig. 1(a)) [11,13].

B. Classification of Memristor-Based Logic

Several memristor-based logic designs have been proposed [7–13]. Fig. 2 classifies them into Boolean, implication, and threshold/majority logic [4]. Note that the logic designs using resistance to represent data are suitable for crossbar-based resistive computing. Two of such designs, resistive Boolean logic and material implication logic, will be discussed next as they are the most popular candidates for crossbars.

C. State-of-the-Art Resistive Logic Types

Resistive Boolean Logic (RBL) provides three primitive gates: NOT, AND, and NAND [13]. It uses high resistance R_H and low resistance R_L to represent a logic 1 and 0, respectively. The two-input NAND gate of Fig. 3(a) is used as an example to explain the working principle of RBL. We assume that the inputs are stored on memristor M_1 and M_2, while the output is produced at M_o. The gate requires an extra resistor R_s ($R_L \ll R_s \ll R_H$). To complete the NAND gate, two steps are performed. First, M_o should be RESET to R_H. Next, control voltages V_h and V_w are applied to the input and output memristors, respectively. In case one of the inputs is 0, the equivalent resistance of M_1 and M_2 is around R_L (see e.g. Fig. 3(a) where M_1=R_L and M_2=R_H). Therefore, the voltage V_x on the nanowire connected to memristors and the resistor is around V_h as $R_L \ll R_s \ll R_H$. As a result, the voltage V_{om} across the output memristor M_o is $V_w - V_x \approx V_w - V_h = \frac{V_w}{2} < V_{ts}$, and therefore M_o stays at R_H (logic 1).

Material Implication Logic (MIL) provides a single primitive gate only, which is material implication (IMP) as shown in Fig. 3(b) [11]. MIL uses R_H and R_L to represent logic 0 and 1, respectively. An IMP gate consists of two memristors M_1 and M_2 and a resistor R_s ($R_L \ll R_s \ll R_H$). M_1 is used as an input, while M_2 is used both as input and output. To perform the IMP operation, the control voltages V_h and V_w should be applied to M_1 and M_2, respectively. In Fig. 3(b), both inputs are assumed to be logic 0 (R_H). After applying V_h and V_w, $V_x \approx 0$ and the output switches to logic 1. Multiple sequential IMP gates can realize AND, OR, or XOR gates [11].

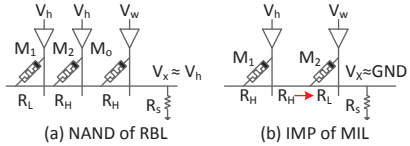


Fig. 3: Logic Designs Suitable for Resistive Computing

Both RBL and MIL have several shortcomings. First, they require several steps to execute a single gate thereby affecting the delay and power. Second, both require additional CMOS controllers to apply the control voltages and control the sequential steps. This impact the performance of the design. Third, both logic designs require the relative high voltage V_w to program the memristors, and hence, each primitive gate consumes more power/energy as compared to having read operations only [3]. Forth, both designs need to switch the output memristors frequently, and therefore, the entire design is strongly limited by the endurance [1,3]. This motivates us to develop scouting logic as described in the next section.

III. SCOUTING LOGIC

This section first describes the main idea of scouting logic. Subsequently, it presents two designs of the sense amplifier used to implement scouting logic.

A. Main Idea

Scouting logic performs its logic operations by modifying the read operation. Fig. 4(a) shows a resistive memory based on 1T1R cells. Normally when a cell is read, say for example Memristor M_1, a read voltage V_r is applied to its row and the switch S_1 is activated. Subsequently, a current I_{in} will flow through the bit line to the input of the sense amplifier (SA). This current is compared to the reference current I_{ref}. If I_{in} is greater than I_{ref} (i.e., when M_1 is R_L state), the output of the SA changes to V_{dd} (logic 1). Similarly, when M_1 is R_H state, $I_{in}<I_{ref}$, and the output changes to logic 0. For proper operations, I_{ref} should be fixed between high and low currents of Fig. 4(b).

Inspired by this read operation, we demonstrate how to implement OR, AND and XOR scouting logic gates, which are frequently used in bitwise logic operations [14,15]. Instead of reading a single memristor at a time, scouting logic activates the two inputs of the gate simultaneously (e.g., M_1 and M_2 in Fig. 4(a)). As a result, the input current to the sense amplifier is determined by the equivalent input resistance ($M_1//M_2$). This resistance results in three possible values: $\frac{R_L}{2}$, $\frac{R_H}{2}$ and $R_L//R_H \approx R_L$. Hence, the input current I_{in} also can have only three values. By changing the value of I_{ref} different gates can be realized. To implement an OR gate, I_{ref} should be fixed between $\frac{2V_r}{R_H}$ and $\frac{V_r}{R_L}$ as depicted in

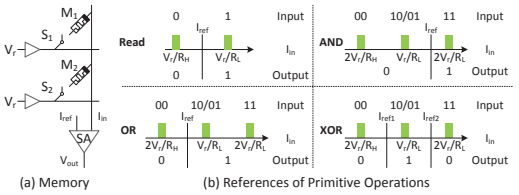


Fig. 4: Main Idea of Scouting Logic

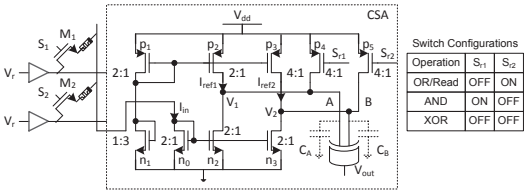


Fig. 5: Current-based SA

Fig. 4(b)). As a result, only when $R_1//R_2=\frac{R_H}{2}$ the output is 0. Similarly, to implement an AND operation, I_{ref} should be fixed between $\frac{2V_r}{R_L}$ and $\frac{V_r}{R_L}$. The XOR operation needs two references and only when $R_1//R_2 \approx R_L$ the output is logic 1. Note that it is also possible to support multi-fanin logic gates by setting proper reference currents.

The above implies that reading logic circuit should satisfy two requirements. First, it should be operational with a single or two references. Second, it should support a reconfigurable reference signal. Different from Pinatubo [7], scouting logic uses sense amplifiers with a lower delay and smaller area. Next, we describe the two SA designs that satisfy the above requirements.

B. Sense Amplifier Design

We propose two SA designs that both satisfy the two requirements: current-based SA (CSA) shown in Fig. 5, and voltage-based SA (VSA) shown in Fig. 6. CSA generates its reference current using transistors p_1 and n_1. This reference current is duplicated via p_2 and p_3 using PMOS current mirrors. Note that the value of I_{ref2} is twice larger than that of I_{ref1} as p_3 has a double size. The input current I_{in} is also mirrored twice through n_2 and n_3. The two pairs of transistors p_2-n_2 and p_3-n_3 implement at the same time two current comparators [16]; they compare the two reference currents (i.e., I_{ref1} and I_{ref2}) with the input current (I_{in}). Transistors p_4 and p_5 determine the logic operation and decide which reference currents are enabled. The table on the right side of Fig. 5 summarizes how p_4 and p_5 are configured for the different gates. For instance, when an XOR is performed both p_4 and p_5 are turned off, i.e. both I_{ref1} and I_{ref2} are considered for the comparison. Assume for this gate that M_1 is R_H and M_2 is R_L. Hence, $I_{ref1}<I_{in}<I_{ref2}$ (see right bottom of Fig. 4(b)). The parasitic capacitor C_A of input A is discharged to ground as $I_{ref1}<I_{in}$, while C_B is charged to V_{dd} as $I_{in}<I_{ref2}$. As a result, the output voltage V_{out} is V_{dd}. The AND and OR gates work in a similar way.

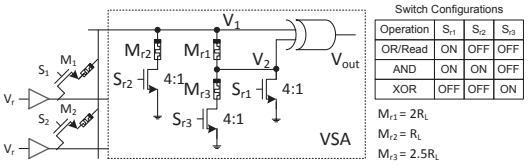


Fig. 6: Voltage-based SA

Fig. 6 shows the VSA. It consists of three reference memristors (M_{ri}, $1\leq i\leq 3$), three switches (S_{ri}, $1\leq i\leq 3$), and an XOR gate. The switches are used to select the reference memristors, while the XOR gate is used as a threshold function. The table on the right part of Fig. 6 shows how the switches are configured for different gates. For instance, to perform an XOR operation, S_{r1} and S_{r2} are turned off while S_{r3} is turned on. After switches S_1 and S_2 are turned on, the input (M_1 and M_2) and the reference memristors form a voltage divider. Let us assume that M_1 is R_H and M_2 is R_L for the XOR gate. By setting M_{r1} and M_{r3} to appropriate values (e.g., $M_{r1}=2R_L$ and $M_{r3}=2.5R_L$), the voltage V_1 will be greater than the threshold voltage of a transistor while V_2 will be less than the threshold voltage. As a result, the output V_{out} approximates 0V. Note that the reference memristors only need to be programed once and may be implemented by resistors.

IV. EVALUATION

This section first verifies scouting logic (SL). Thereafter, it provides a comparison with state-of-the-art.

A. Design Verification

The SL gate simulations are verified with Cadence Spectre. The simulation model consists of the 1T1R array of Fig. 4(a) connected to either CSA or VSA; both SAs have been verified. The 1T1R array and SAs are described by a SPICE netlist, while the memristor model [17], CMOS controller and voltage drivers by Verilog-A modules. The simulation parameters are extracted from references [11,13,18] and summarized in Table I. The transistor sizes of the CSA and VSA designs are depicted in Figs. 5 and 6, respectively and are simulated with the PTM 90nm [19] library. Here, we assume that all the memristors already store same data (in 1T1R array).

The AND, OR and XOR gates are verified for all the possible memristor states. Fig. 7(a) and (b) show an example of waveforms for the XOR gate based on CSA and VSA, respectively. Here, the states M_1=R_H (logic 0) and M_2=R_L (logic 1) are simulated. To evaluate the output of the XOR gate, the switches S_1 and S_2 of Fig. 5 should be turned on to read the memristors M_1 and M_2. To configure the CSA of Fig. 5 as an XOR gate, the switches S_{r1} and S_{r2} should be turned off. As a result, the voltage of node V_1 of CSA is 0.256V$<V_{th,cmos}$=0.45V while the voltage of node V_2 is 0.59V$>V_{th,cmos}$=0.45V (see also Fig. 5). The final output V_{out} of the XOR gate is 0.894V (logic 1). VSA-based XOR gate

TABLE I: Parameters

Parameter	Description	Value
Technology		
Memristor (TaO_x) [1,18]		
F (nm)	Feature size	90
V_{th} (V)	Threshold voltage for SET	1.17
V_{tr} (V)	Threshold voltage for RESET	1.06
R_{L} ($k\Omega$)	Low resistance	200
R_{H} ($M\Omega$)	High resistance	10
A_{cell} (μm^2)	Area of a 1T1R cell	0.0486
T_{sw}(ns)	Switching time (max of SET and RESET)	1.71
CMOS		
UMC 90nm Library		
Design [11,13]		
N_{R}	No. of rows in 1T1R array	128
N_{C}	No. of columns in 1T1R array	32
V_{w} (V)	Program voltage	1.6
V_{h} (V)	Half-select voltage	0.8
V_{r} (V)	Read voltage	0.9
V_{dd} (V)	CMOS power supply	0.9

works in a similar way as shown in Fig. 7(b). Note that the VSA based design is faster than the CSA design.

B. Comparison with the State-of-the-art

SL gates (i.e., AND, OR and XOR gates) both based on CSA (SL_CSA) and on VSA (SL_VSA) are compared to RBL and MIL gates in terms of delay, power consumption and area using a 128x32 1T1R memory array. The considered components are the 1T1R memory array, the CMOS controller and SAs. For RBL and MIL, additional memristors required to store intermediate results and implement R_s as shown in Fig. 3 are also considered. The delay and power consumption of the 1T1R memory array are obtained from Cadence Spectre while the area of a single 1T1R cell is taken from ITRS [1]. CMOS controllers are synthesized and evaluated using Cadence RTL Compiler with UMC 90nm library. The delay and power consumption of the SAs are obtained from Cadence Spectre while the area of a single SA is obtained from Cadence Virtuoso using Cadence 90nm PDK [20]. For the modified SA used by SL, only the additional transistors are considered as compared to the current SA of [21].

Fig. 8 presents the results of the comparison:

- **Delay**: CSA-based SL has the lowest delay to execute a single step as well as the lowest total delay (see Fig. 8(a) and (b)); its total delay is at least 2.48 times shorter than RBL and MIL. Although RBL and MIL have a shorter delay per step than VSA-based SL, their total delay is longer as they need multiple steps to execute a logic operation.
- **Power**: VSA-based SL consumes the lowest power for all the gates; it consumes at least 2.36 times less power than RBL and MIL as it can execute a logic operation in a single step. In addition, the controller dominates the power consumed by RBL and MIL, while the SAs dominate the power consumption for SL.
- **Area**: SL does not need more area than RBL and MIL. Although CSA- and VSA-based SL need additional area for the modified SA, it utilizes a simple CMOS controller. In contrast, RBL and MIL require additional area for their more complex controller as they need several steps to execute the gate. The total additional area of the extra cells for MIL and RBL is negligible.

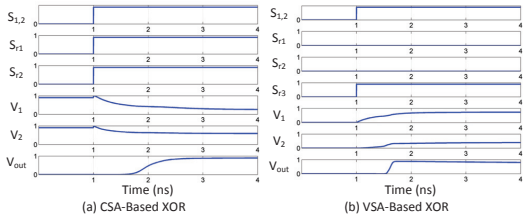


Fig. 7: SPICE Simulation Results

The comparison results clearly show that SL outperforms RBL and MIL in terms of delay and power consumption while having the same or less area overhead.

Fig. 9 compares SL with Pinatubo [7] in terms of delay and area. However, as the authors of [7] provided a design without implementation details (i.e., $\frac{W}{L}$ ratio of the transistors), we use the number of computation steps as the delay metric and the number of required transistors and memristors as the area metric. Compared to Pinatubo, SL requires only one step to execute the XOR gate. All other gates can be executed in a single step. In addition, the current- and voltage-based sense amplifiers require much less transistors (40% and 64%, respectively). Therefore, SL is potentially faster with a lower area overhead than Pinatubo.

V. DISCUSSION

This section discusses potential applications and some challenges and solutions of scouting logic design.

A. Potential Applications

SL can implement many bitwise logic operations, such as AND, OR and XOR, in a very efficient manner as pointed out earlier. Such logic operations are frequently used in many data-intensive applications such as database queries [15], graph processing [14], etc. In traditional settings, these applications need to transfer data between the processor and memory to perform these bitwise logic operations. The movement of such large amount of data input/output of the memory results in considerable amount delay and energy overhead [14,15]. Since SL can directly perform logic operations within the memory array, it can eliminate the need for data movement between processors and memory, thus, significantly reducing execution time and energy consumption. Moreover, scouting logic based memories have very good scaling characteristics due to the fact that only memory controller need to be adapted.

B. Challenges

Memristor technology has been being extensively studied for the past few years due to its applicability in logic and memory designs [4]. Memristor devices have been implemented using different material and cell structures, but nevertheless all suffer from the same two major challenges, namely, limited cell endurance and device variability [4,22]. Performing logic operations using scouting logic requires only reading memristors' states as compared to other memristor-based logic designs that require switching between resistive states.

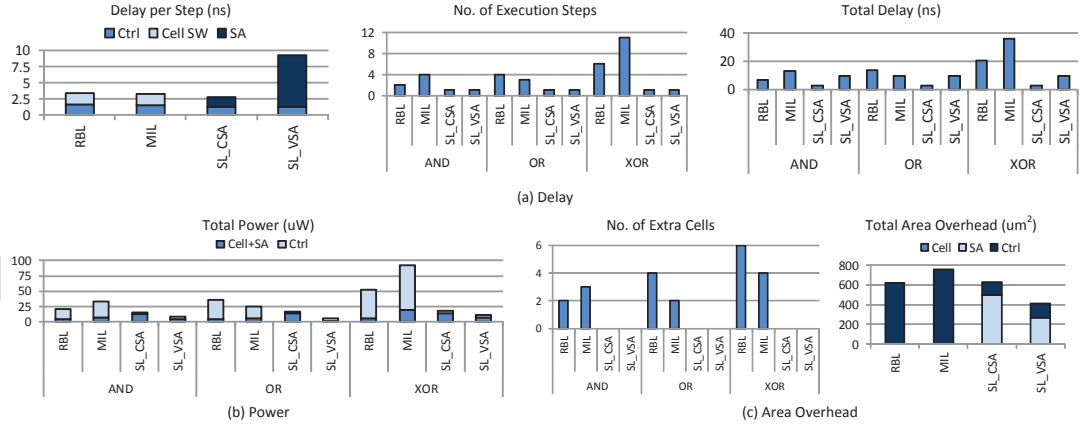


Fig. 8: Comparison Between Different Logic Styles

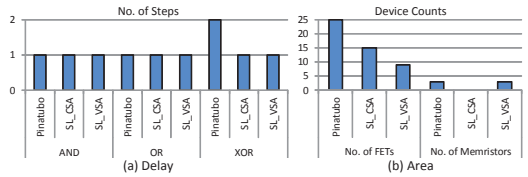


Fig. 9: Comparison with Pinatubo

Requiring only read operations makes scouting logic designs have better endurance as compared to other approaches resulting in improved device lifetime.

Similar to all memristor based logic, scouting logic suffers from CMOS (i.e. transistor mismatch [23]) and memristor variations (cycle-to-cycle and device-to-device [22]). Such variations may cause circuit failures if not addressed appropriately. Extensive research on CMOS process variations in sense amplifiers has been proposed [23,24]; hence we will elaborate only on memristor variations and how scouting logic can be made more resilient to such variations.

Fig. 10 shows a methodology that can be used to realize a more robust scouting logic design against variations. Firstly, the design is simulated without variations and the range for R_L and R_H are determined. Fig. 11(a) shows the relationship between the equivalent resistance of two input memristors (R_{eq}) and the normalized output voltage ($\frac{V_{out}}{V_{dd}}$) of the sense amplifier of Fig. 5 configured for XOR operation; we assume here that the output threshold for logic 1 is $0.6V_{dd}$ and $0.4V_{dd}$ for logic 0 (see red lines in Fig. 11(a)). Hence, the resistance ranges for R_L and R_H can be determined by sweeping the resistance values of memristors using SPICE simulations.

Since process variations may cause the resistance values to deviate from their intended values [22,25], the equivalent resistance might fall outside the working range of the memristor leading to operational failure. Variations are typically modeled as a normal distribution as shown in Fig. 11(b) [22], where the

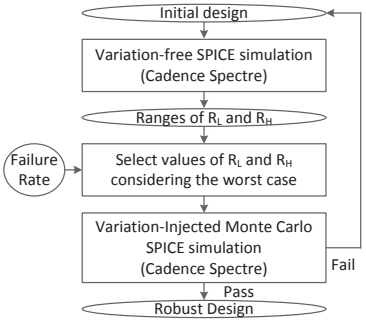


Fig. 10: Variation Resilient Design Methodology

normalized standard deviation versus the mean (or intended) resistance values is given. The figure shows that R_H suffers from large variance as compared to R_L, and that the standard deviation reduces when smaller values for R_L and R_H are used. It is possible to use the standard deviation to express the failure rate. To realize a failure rate of e.g., 10^{-3}, the value of R_L or R_H should fall in the range of n_σ=3σ of the normal distribution [24] (i.e. μ+3σ and μ−3σ). For example, a robust XOR gate design considering the variability should satisfy the following equations:

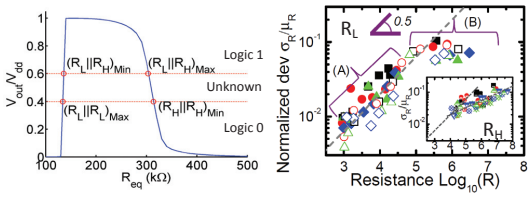
$$\frac{1}{2}(R_L + n_\sigma\sigma_{R_L}) < (R_L||R_L)_{\text{Max}} \quad (1)$$

$$\left[(R_L - n_\sigma\sigma_{R_L})\right]||\left[(R_H - n_\sigma\sigma_{R_H})\right] > (R_L||R_H)_{\text{Min}} \quad (2)$$

$$\left[(R_L + n_\sigma\sigma_{R_L})\right]||\left[(R_H + n_\sigma\sigma_{R_H})\right] < (R_L||R_H)_{\text{Max}} \quad (3)$$

$$\frac{1}{2}(R_H - n_\sigma\sigma_{R_H}) > (R_H||R_H)_{\text{Min}} \quad (4)$$

where σ_{R_L} and σ_{R_H} represent the standard deviation of R_L and R_H, respectively. Eq. 1 applies when both XOR inputs are 1, and hence output is 0. As the maximum value of the low resistance of each input memristor is $R_L + n_\sigma\sigma_{R_L}$, the maximum equivalent resistance $\frac{1}{2}(R_L + n_\sigma\sigma_{R_L})$ should be



(a) Relation between Resistance and Output Voltage

(b) Normalized Deviation of Resistance [15]

Fig. 11: Range and Normalized Deviation of Resistance

less than $(R_L||R_L)_{\text{Max}}$. Eq. 2 and 3 are the constraints for inputs 01/10 while Eq. 2 for inputs 00.

Once R_L and R_H are defined, the design is subsequently simulated using Monte Carlo simulations in Cadence Spectre [20]. The total number of failed simulations are monitored. If failure rate is lower than the requirement, then the design is considered robust; otherwise, the whole process is restarted by selecting new reference currents (i.e. redesign). Designs can be modified in two ways; either to modify mean resistance values, or tune the reference current/voltage value of the sense amplifier (i.e., $\frac{W}{L}$ of transistors in CSA or M_i of VSA). Changing mean resistances values can be accomplished by the writing circuity; e.g., by tuning the voltage pulse duration or amplitude to map logic states to resistance states as needed.

To verify the robust design methodology of Fig. 10, we apply the proposed approach to CSA-based scouting logic approach as a case study. Here, a 3σ design is used which satisfies a failure rate of 10^{-3}. The initial design (input of Fig. 10) is compared with the robust design (output of Fig. 10) in terms of failure rate. To verify the proposed approach, a 1000-iteration Monte Carlo simulation is conducted using Cadence Spectre simulator and the output voltage for each simulation run is monitored. The output voltage for logic 1 requires a voltage of $0.6\times V_{\text{dd}}$ or higher while for a logic 0 should be below $0.4\times V_{\text{dd}}$. Therefore, if a simulation run results in an output that does not fall in either range will be considered as an unknown and hence unreliable operation. If the logic state of the simulated output is different from the expected one, the design fails; otherwise, it passes. Table II summarizes technology and Monte Carlo simulation parameters used in the experiments. Note that the σ of the normal distribution are extracted from [22] depending on the selected resistance value.

The bottom of Table II shows failure rates as given by the Monte Carlo simulations. For the OR gate, both the initial and the robust design versions pass. However, for AND and XOR gates, the initial version fails when logic states are 01/10 while the robust version still passes. Therefore, the methodology proposed in Fig. 10 can enhance the robustness of the design to deal with resistance variation in scouting logic based design.

VI. CONCLUSION

This paper proposed scouting logic design for resistive computing. Such design does not reduce device endurance. In addition, it outperforms the existing memristor-based logics in terms of delay and power consumption, while using similar or less area.

TABLE II: Parameters and Results of Monte Carlo Simulations

Technology Parameters			
MOSFET	$p_{1,2}$	$p_{3,4,5}$	$n_{0,1,2,3}$
$\frac{W}{L}$	4	8	4
Version	Initial		Robust
R_H/R_L	50		
R_L(kΩ)	23		30
R_H(kΩ)	1150		1500

Monte Carlo Simulation Parameters		
Iteration	1000	
Version	Initial	Robust
σ_{R_L}/μ_{R_L} [22]	0.0261	0.0281
σ_{R_H}/μ_{R_H} [22]	0.0406	0.0418
V_{H}	$0.6\times V_{\text{dd}}$	
V_{L}	$0.4\times V_{\text{dd}}$	

Failure Rate						
Version	Initial			Robust		
Gate \ Input	00	01/10	11	00	01/10	11
AND	0	19.10%	0	0	0	0
OR	0	0	0	0	0	0
XOR	0	45%	0	0	0	0

REFERENCES

[1] ITRS ERD report. [Online]. Available: http://www.itrs.net
[2] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*, 2012.
[3] J. J. Yang *et al.*, "Memristive devices for computing," *Nat. Nano*, 2013.
[4] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *DATE*. IEEE, 2017.
[5] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*. IEEE, 2015.
[6] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *NANOARCH*. IEEE, 2015.
[7] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.
[8] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, 2012.
[9] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *TNANO*, 2013.
[10] E. Linn *et al.*, "Beyond von neumannlogic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, 2012.
[11] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.
[12] S. Kvatinsky *et al.*, "Mrl: memristor ratioed logic," in *CNNA*. IEEE, 2012.
[13] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *DTIS*. IEEE, 2016.
[14] V. Agarwal *et al.*, "Scalable graph exploration on multicore processors," in *SC*. ACM, 2010.
[15] J. Chou *et al.*, "Parallel index and query for large scale data analysis," in *SC*. ACM, 2011.
[16] L. Samuel, "Cmos current comparator with regenerative property," *IJECSE*, 2013.
[17] A. Siemon *et al.*, "Simulation of tao x-based complementary resistive switches by a physics-based memristive model," in *ISCAS*. IEEE, 2014.
[18] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Adv. Mat.*, 2011.
[19] Predictive transistor model. [Online]. Available: http://ptm.asu.edu/
[20] Cadence ic design kit. [Online]. Available: https://www.cadence.com/
[21] M.-F. Chang *et al.*, "An offset-tolerant fast current-sampling-based sense amplifier for small-cell-current nonvolatile memory," *JSSC*, 2013.
[22] A. Fantini *et al.*, "Intrinsic switching variability in hfo 2 rram," in *IMW*. IEEE, 2013.
[23] W. Dehaene *et al.*, "Variability-aware design of low power sram memories," 2009.
[24] I. Agbo *et al.*, "Quantification of sense amplifier offset voltage degradation due to zero-and run-time variability," in *ISVLSI*. IEEE, 2016.
[25] M. Hu *et al.*, "Geometry variations analysis of tio 2 thin-film and spintronic memristors," in *ASP-DAC*. IEEE, 2011.

On the Robustness of Memristor Based Logic Gates

Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Mottaqiallah Taouil, Said Hamdioui
Laboratory of Computer Engineering, Delft University of Technology, Delft, the Netherlands
Email: {L.Xie,H.A.DuNguyen,J.Yu-1,M.Taouil,S.Hamdioui}@tudelft.nl

***Abstract*—As today's CMOS technology is scaling down to its physical limits, it suffers from major challenges such as increased leakage power and reduced reliability. Novel technologies, such as memristors, nanotube, and graphene transistors, are under research as alternatives. Among these technologies, memristor is a promising candidate due to its great scalability, high integration density and near-zero standby power. However, memristor-based logic circuits are facing robustness challenges mainly due to improper values of design parameters (e.g., OFF/ON ratio, control voltages). Moreover, process variation, sneak path currents and parasitic resistance of nanowires also impact the robustness. To realize a robust design, this paper formulates proper constraints for design parameters to guarantee correct functionality of logic gates (e.g., AND). Our proposal is verified with SPICE simulations while taking both device variation and parasitic effects into account. It is observed that the errors due to analytical parameter constraints are typically within 4.5% as compared to simulations.**

I. INTRODUCTION

As today's CMOS technology is scaling down to its physical limits, it suffers from major challenges such as saturated performance gain, increased leakage power, and reduced reliability. Emerging technologies (such as nanotube transistors, graphene transistors, magnetic tunneling junctions and memristors) are being investigated as alternatives [1–3]. Among these technologies, memristor is a promising candidate due to its great scalability, near-zero standby power and compatibility with CMOS process [1,2].

Many potential applications based on memristor technology have been proposed, such as non-volatile memory, neuromorphic circuits and resistive computing architectures, and logic [2]. With respect to logic, three types of memristor-based logic have been proposed;namely threshold/majority, implication, and Boolean logic [2]. In threshold/majority logic, memristors are used as input weights while CMOS current mirrors or inverters implement the threshold function. Both implication and Boolean logic use resistance to represent the data and therefore can be easily integrated in high density memory based on crossbar architectures [4,5]. However, implication and Boolean logic are facing robustness issues, such as unexpected switching of memristors, which leads to an erroneous functionality. These issues are typically caused by *improper design parameters* and thus, proper constraints must be formulated to realize a robust design.

Only limited work has reported such design constraints. In [6], Kvatinsky et al. reported a set of constraints for individual implication logic gates. However, this work did not consider device variation [7], sneak path currents [4], and nanowire resistances. In [4], Zhu et al. reported a methodology to derive design constraints while considering sneak path currents within the crossbar. However, they did not consider dynamic switching process of memristors and other effects such as device variation [7] and nanowire resistances. Both [6] and [4] focused on implication logic and no such work has been done for Boolean logic. More importantly, the impact of device variation, dynamic switching process, sneak path currents, and nanowire resistances on robustness have not been intensively studied.

To address above issues, this paper proposes a novel methodology to analytically formulate design constraints, while considering the dynamic switching of memristors. The impact of device variation and parasitics effects as well as the impact of different memristor models are studied extensively using simulations. To illustrate our proposal, this paper uses resistive Boolean logic [5,8] as an example. Note that our approach also can be applied to implication logic.

The rest of this paper is organized as follows. Section II describes the fundamentals of the resistive Boolean logic. Section III illustrates the proposed approach. Section IV verifies our approach using SPICE simulations. Finally, section V concludes the paper.

II. FUNDAMENTALS OF RESISTIVE BOOLEAN LOGIC

This section first explains the memristive behaviour. Thereafter, it classifies reported memristor models and describes the model [7] used in this paper. Finally, it presents the primitive logic gates of resistive Boolean logic (RBL) [5].

A. Memristive Behaviour

Fig. 1 shows the I-V relation of a memristor, which has a high (R_H) and low (R_L) resistance [10]. Initially, when a memristor is fabricated, it cannot switch until a high voltage V_f is applied to the memristor, referred to as forming process [10]. Thereafter, the memristor switches from one resistive state to another in case the absolute value of the voltage across the device is greater than its threshold voltage.

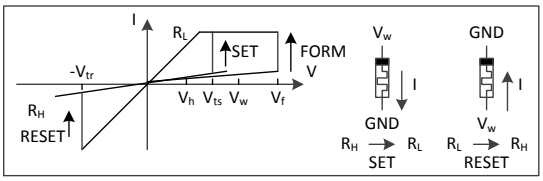


Fig. 1: Behaviour of a Memristor [9].

978-1-5386-0473-1/17/$31.00 © 2017 IEEE

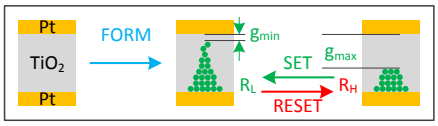


Fig. 2: Mechanism of Resistive Switching

Otherwise, it stays in its current resistive state. A memristor typically requires two different threshold voltages to switch from high-to-low (SET, V_{ts}) and from low-to-high (RESET, V_{tr}<V_{ts}) resistance [7]. In the figure, the black squares at the edges of the memristors represent their positive terminal. RBL uses voltages V_w and V_h to control its primitive operations. V_w is used to program memristors. V_h is used to (i) implement gates (e.g., NAND), (ii) minimize sneak path currents and (iii) avoid unexpected programming [5]. The typical relationship between control and threshold voltages is valuing $0<V_h=\frac{V_w}{2}<V_{tr}<V_{ts}<V_w<2V_{tr}$, which provides a rough estimate of a valid voltage range [8]. A more accurate range will be shown later in Section IV.

Fig. 2 shows the resistive switching process [10]. After the forming process, a conductive filament consisting of oxygen vacancies is created in the TiO_2 layer. The gap g between the top electrode and the tip of the conductive filament determines the resistive state [10]. When the gap is at its minimum (i.e., g_{min}), the memristor is in R_L state. In contrast, when the gap is at its maximum (i.e., g_{max}), the memristor is in R_H state. This gap g is tuned by applying the voltage as shown in Fig. 1. For instance, when V_w>V_{ts} and *GND* are applied to the top and bottom terminal of the device respectively, the gap is reduced to g_{min} and the memristor is SET to R_L.

B. Memristor Models and its Classification

To formulate the memristive behaviour, many memristor models have been proposed [11,12]. These models can be classified using the following two criteria [11,12]:

- Switching process: it describes whether the memristor switches *abruptly* or *smoothly* when the voltage across the memristor is greater than its threshold voltage.
- I-V relation: it describes whether the current I flowing through the memristor is a *linear* (i.e., $I=\frac{V}{R}$, R is the resistance of memristor) or *nonlinear* (i.e., $I=f(V)$, where f is a nonlinear function) function of the voltage V across it.

Applying the above two criteria, memristor models are classified into three classes:

- Ideal models: this class of models apply abrupt switching and their I-V relation is linear, such as the behavioural model of Fig. 1.
- Linear models: this class of models apply smooth switching and their I-V relation is linear, such as Biolek's model [13].
- Nonlinear models: this class of models apply smooth switching and their I-V relation is nonlinear, such as Guan's model [7].

To illustrate our proposal, this paper uses Guan's model as an example, since it is a calibrated model that can also model

TABLE I: Parameters of Guan's model [7]

Parameter	Description	Value
ν_0 (nm/ns)	Velocity of ions	10
E_a (eV)	Activation energy barrier	0.6
a (m)	Hopping distance of ions	2.50E-10
F_{min} (V/m)	Minimum field requirement	1.40E+09
q (C)	Charge value of a single electron	1.60E-19
k ($m^2kgs^{-2}K^{-1}$)	Boltzmann constant	1.38E-23
T (K)	Temperature	298
t_{ox} (m)	Thickness of the oxide layer	1.20E-08
g_{min} (m)	Minimum of the gap	2.00E-10
g_{max} (m)	Maximum of the gap	2.10E-09
γ_0	Fitting parameters	16
β	Fitting parameters	0.8
I_0 (A)	Fitting parameters	1.00E-03
g_0 (m)	Fitting parameters	2.50E-10
V_0 (V)	Fitting parameters	0.25

device variation. Guan's model formulates the I-V relation using a tunnelling current, which is tuned by the voltage V across the device and the gap g. This I-V relation is expressed in Eq. 1 [7]:

$$I(g,V) = I_0 \cdot \exp\left(-\frac{g}{g_0}\right) \cdot \sinh\left(\frac{V}{V_0}\right) \quad (1)$$

where I_0, g_0 and V_0 are fitting parameters which depend on different manufacturing configurations (e.g., materials).

The gap change rate is modelled by the probability that oxygen ions overcome the activation energy barriers E_a; this change rate is expressed in Eq. 2 [7]:

$$\frac{dg}{dt} = \nu_0 \cdot \exp\left(-\frac{E_a}{kT}\right) \cdot \sinh\left(\frac{qa\gamma V}{t_{\text{ox}}kT}\right) \quad (2)$$

where ν_0 is the velocity, E_a the activation energy barrier, k the Boltzmann constant, T the temperature of the device, q the charge value of a single electron, a is the hopping distance of the ions (e.g., oxygen ions), t_{ox} the thickness of the oxide layer (e.g., TiO_2), and $\gamma = \gamma_0 - \beta g$ is the local enhancement factor (γ_0 and β are fitting parameters). Table I summarizes the values of the model parameters.

To start resistive switching, the voltage V across the device should satisfy the condition in Eq. 3 [7]:

$$V > \frac{F_{\text{min}} t_{\text{ox}}}{\gamma_0 - \beta g} \quad (3)$$

where F_{min} is the minimum required field to enhance the gap formation. Note that this models the different threshold voltages for SET and RESET (see Fig. 1).

C. Primitive Gates

RBL uses AND and NAND as primitive gates [5]. As AND and NAND gates work in a similar way, the AND gate of Fig. 3 is used as an example to illustrate its working principle. Both the fan-in and fan-out of this AND gate are two; two input and output memristors are employed. The high and low resistance of a memristor are used to represent logic 1 and 0, respectively. The triangles in the subfigures represent voltage drivers. Performing an AND gate needs three steps as shown in Fig. 3(a)–(c), respectively. Let us assume that all inputs are

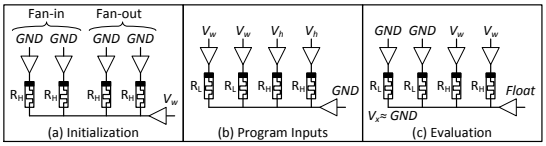


Fig. 3: An Example of an AND Gate

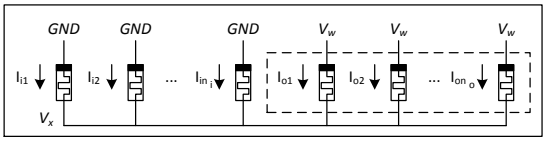


Fig. 4: Generic AND Gate in Evaluation Step

logic 0. First, all the input and output memristors are initialized to R_H by applying V_w and *GND* to the row and columns, respectively (see Fig. 3(a)). Second, all input memristors are SET to R_L as both inputs are logic 0 (see Fig. 3(b)); *GND* is applied to the row while V_w is applied to columns of input memristors. To avoid programming the output memristors, V_h is applied to the columns of output memristors. Finally, the AND gate is evaluated by applying *GND* and V_w to the columns of input and output memristors, respectively (see Fig. 3(c)); while the row is kept floating. Hence, the voltage V_x of the floating row is around 0 as $R_H \gg R_L$, and the voltage across the output memristor $V_{om}=V_w-V_x \approx V_w > V_{ts}$. As a result, the two output memristors switch to R_L. The same reasoning can be applied to other input combinations.

III. METHODOLOGY OF FORMULATING CONSTRAINTS

To determine the values of the design parameters appropriately, a set of constraints need to be defined. This section first defines how the constraints are derived. Thereafter, the method is applied to an AND gate as a case study. Finally, it discusses the factors that impact the parameter constraints.

A. Methodology

The AND gate of Fig. 3 functions correctly when all the memristors of the gates should switch correctly and on the right time. This correct switching can be formulated by relationships describing the voltage across the memristors and their threshold voltages; they are referred to as switching conditions. Motivated by this, a three-step methodology is proposed as follows:

- **Step 1** Formulate the unknown voltages of the floating nanowires (e.g., V_x of Fig. 3(c)), as they are required to obtain the voltage across the input and output memristors. To formulate them, Kirchhoff's circuit law is applied to the logic gates.
- **Step 2** Predict the resistance of the output memristors. As the resistive switching is a dynamic process, output memristors will stop switching once the voltage across them do not satisfy the minimum voltage requirement (e.g., Eq. 3).
- **Step 3** Formulate the parameter constraints by combining the switching conditions of input and output memristors.

To formulate the voltages of the floating nanowires, the following assumptions are made.

- The resistance of nanowires is initially ignored. Its impact will be studied later in Section IV using SPICE simulations. The capacitance of the nanowires is neglected as it mainly impacts the delay [14].
- The device variation is initially ignored. Its impact will also be studied in Section IV using SPICE simulations.

B. Case Study: AND Gate

To illustrate the proposed methodology, it is applied to an AND gate with n_i fan-in and n_o fan-out. Fig. 4 shows this AND gate, which consists of n_i input and n_o output memristors. The only relevant step in Fig. 3 where the switching conditions have to be checked is the evaluation step, as it is the only case where the row nanowire is floating and unexpected or wrong switching may occur.

Step 1 To formulate the unknown voltage V_x of the floating nanowire, Kirchhoff's current law is applied to the AND gate. The result is expressed in Eq. 4.

$$\sum_{p=1}^{n_i} I_{\text{ip}} + \sum_{q=1}^{n_o} I_{\text{oq}} = 0 \tag{4}$$

where $I_{\text{ip}}(1 \le p \le n_i)$ and I_{oq} $(1 \le q \le n_o)$ are the currents flowing through the input and output memristors, respectively. As the voltage across all the input memristors equals $0-V_x$, while the voltage across the output memristors V_w-V_x. Substituting I_{ip} and I_{oq} in Eq. 4 with Eq. 1 results in:

$$\sum_{p=1}^{n_i} \left[I_0 \exp\left(-\frac{g_{\text{ip}}}{g_0}\right) \sinh\left(\frac{-V_x}{V_0}\right) \right] + \sum_{q=1}^{n_o} \left[I_0 \exp\left(-\frac{g_{\text{oq}}}{g_0}\right) \sinh\left(\frac{V_w - V_x}{V_0}\right) \right] = 0 \tag{5}$$

where g_{ip} $(1 \le p \le n_i)$ presents the gap of the input memristors, g_{oq} $(1 \le q \le n_o)$ the gap of output memristors. Let us assume the number of low-resistance input memristors is n_{L} and their gaps are g_{min}. Therefore, the remaining $n_{\text{i}} - n_{\text{L}}$ input memristors are in high resistance; their gaps are g_{max}. Meanwhile, all output memristors have the gap g_{out}. As a result, Eq. 5 is rewritten as

$$n_{\text{L}} I_0 \exp\left(-\frac{g_{\text{min}}}{g_0}\right) \sinh\left(\frac{-V_x}{V_0}\right) + (n_{\text{i}} - n_{\text{L}}) I_0 \exp\left(-\frac{g_{\text{max}}}{g_0}\right) \sinh\left(\frac{-V_x}{V_0}\right) + n_{\text{o}} I_0 \exp\left(-\frac{g_{\text{om}}}{g_0}\right) \sinh\left(\frac{V_w - V_x}{V_0}\right) = 0 \tag{6}$$

To obtain the expression of V_x, Eq. 6 is solved as following

$$V_x = \frac{V_0}{2} \ln \left[\frac{a + n_{\text{o}} \exp\left(\frac{V_w}{V_0} - \frac{g_{\text{om}} - g_{\text{min}}}{g_0}\right)}{a + n_{\text{o}} \exp\left(-\frac{V_w}{V_0} - \frac{g_{\text{om}} - g_{\text{min}}}{g_0}\right)} \right] \tag{7}$$

where a is

$$a = n_{\text{L}} + (n_{\text{i}} - n_{\text{L}}) \exp\left(\frac{g_{\text{min}} - g_{\text{max}}}{g_0}\right) \tag{8}$$

Step 2 This step predicts the resistance of the output memristors. When all inputs are in high resistance (i.e., logic 1), the output memristors should stay at R_H. As the output memristors are already initialized to R_H (see Fig. 3(a)), it is not necessary to predict the resistance of output memristors. In contrast, when at least one of input memristors (i.e., n_{L}>0) is in low resistance, the output memristors must switch from high to low resistance. During this switching process, V_x increases starting from 0V. Therefore, the voltage across the output memristors (i.e., V_w−V_x) decreases. Once this voltage cannot satisfy the condition of Eq. 3, the output memristors stop switching. At this moment, V_x and the gap of output memristors g_{om} satisfy the following condition:

$$V_w - V_x = \frac{F_{\mathrm{min}} t_{\mathrm{ox}}}{\gamma_0 - \beta g_{\mathrm{om}}} \tag{9}$$

To obtain V_x and g_{om}, Eq. 7 and 9 need to be solved. As it is complex to solve Eq. 7, it will be first simplified. Typically, $\exp\left(\frac{g_{\mathrm{max}}-g_{\mathrm{min}}}{g_0}\right)$ and $\exp\left(\frac{V_w}{V_0} - \frac{g_{\mathrm{om}}-g_{\mathrm{min}}}{g_0}\right)$ are greater than 1000 [7]. Hence, $\exp\left(\frac{g_{\mathrm{max}}-g_{\mathrm{min}}}{g_0}\right) \gg \frac{n_i - n_{\mathrm{L}}}{n_{\mathrm{L}}}$ and $\exp\left(\frac{V_w}{V_0} - \frac{g_{\mathrm{om}}-g_{\mathrm{min}}}{g_0}\right) \gg \frac{n_o}{n_{\mathrm{L}}}$ when the number of inputs and outputs are not very large (e.g., 100). As a result, Eq. 7 is approximated as follows:

$$\begin{aligned} V_x &\approx \frac{V_0}{2} \ln\left[\frac{n_{\mathrm{o}} \exp\left(\frac{V_w}{V_0} - \frac{g_{\mathrm{om}}-g_{\mathrm{min}}}{g_0}\right)}{n_{\mathrm{L}}}\right] \\ &= \frac{V_0}{2}\left[\ln\left(\frac{n_{\mathrm{o}}}{n_{\mathrm{L}}}\right) + \frac{V_w}{V_0} - \frac{g_{\mathrm{om}} - g_{\mathrm{min}}}{g_0}\right] \end{aligned} \tag{10}$$

Solving Eq. 9 and 10 provides the approximated values for V_x and g_{om}; they are

$$V_x = \frac{V_0}{4}\left[\frac{g_{\mathrm{min}}}{g_0} - \ln\left(\frac{n_{\mathrm{o}}}{n_{\mathrm{L}}}\right) + 3\frac{V_w}{V_0} - \frac{\gamma_0}{\beta g_0} + \sqrt{\Delta}\right] \tag{11}$$

$$g_{\mathrm{om}} = \frac{g_0}{2}\left[\frac{g_{\mathrm{min}}}{g_0} - \ln\left(\frac{n_{\mathrm{o}}}{n_{\mathrm{L}}}\right) - \frac{V_w}{V_0} + \frac{\gamma_0}{\beta g_0} - \sqrt{\Delta}\right] \tag{12}$$

where Δ is

$$\Delta = \left[\frac{g_{\mathrm{min}}}{g_0} - \ln\left(\frac{n_{\mathrm{o}}}{n_{\mathrm{L}}}\right) - \frac{V_w}{V_0} - \frac{\gamma_0}{\beta g_0}\right]^2 - \frac{8 F_{\mathrm{min}} t_{\mathrm{ox}}}{\beta V_0 g_0} \tag{13}$$

Step 3 To formulate the parameter constraints, the switching behaviour of the input and output memristors need to be identified first. Thereafter, we express the conditions based on the expected switching behaviour of each memristor. These conditions are used to derive the constraints.

We use the case where all inputs are logic 1 (i.e., n_{L}=0) as an example. Therefore, in the evaluation step, all input and output memristors should stay at R_H. The voltage across the input memristors is V_x−V_w and it is always negative. Therefore, input memristors always stay at R_H (see also Fig. 1). In contrast, to guarantee that the output memristors stay at R_H, the voltage across them (i.e., $V_w - V_x < V_{\mathrm{ts}}$) should be less than V_{ts}. This condition is expressed by

$$V_w - V_x < V_{\mathrm{ts}} = \frac{F_{\mathrm{min}} t_{\mathrm{ox}}}{\gamma_0 - \beta g_{\mathrm{max}}} \tag{14}$$

Therefore, all design parameters (e.g., n_i, n_o, V_w) should satisfy the constraint expressed in Eq. 14. By substituting n_{L} with 0 in Eq. 7, V_x is expressed as

$$V_x = \frac{V_0}{2} \ln\left[\frac{n_{\mathrm{i}} \exp\left(\frac{g_{\mathrm{min}}-g_{\mathrm{max}}}{g_0}\right) + n_{\mathrm{o}} \exp\left(\frac{V_w}{V_0} - \frac{g_{\mathrm{max}}-g_{\mathrm{min}}}{g_0}\right)}{n_{\mathrm{i}} \exp\left(\frac{g_{\mathrm{min}}-g_{\mathrm{max}}}{g_0}\right) + n_{\mathrm{o}} \exp\left(-\frac{V_w}{V_0} - \frac{g_{\mathrm{max}}-g_{\mathrm{min}}}{g_0}\right)}\right] \tag{15}$$

The same process can be applied to the case where at least one input is 0. Note that the constraints should distinguish between the following two cases: before (to make sure that the switching condition is right) and after (to make sure that the memristors do not change back) the output memristors switch. The complete constraints can be obtained by applying the three-step methodology to all possible cases. It is worth to note that the constraints mainly put restrictions on the values of the control voltages V_w and V_h, since the other parameters are determined by technology (e.g., g_{max}, g_{min}) and gate configurations (e.g., n_i, n_o).

C. Factors Impacting the Accuracy of Constraints

The accuracy of the obtained constraints is impacted by the following factors, which are ignored in the formulation:

1) *Device variation*. The fabricated memristors suffer from device variations [10]. Guan's model formulates the device variation by the variations in the gap, which determines the resistance of the memristors. Hence, it impacts the voltage of the floating nanowire and the voltage across memristors.
2) *Resistance of nanowires*. Nanowires have a parasitic resistance. It induces a voltage drop along the nanowires, and therefore the voltage across memristors along the nanowires may change. Note that the capacitance of nanowires mainly impacts the delay of the logic circuits [14].
3) *Sneak path currents within crossbar*. When multiple RBL gates are mapped on the crossbar, sneak path currents will induce voltage drifts on floating nanowires [8]. These voltage drifts can disturb the gates or even cause failures [6,8].
4) *Types of memristor models*. Different types of memristor models (e.g., ideal, linear or non-linear models [11,12]) use a different I-V relation and switching process. They impact the calculated values of V_x and output resistance after switching.

The impact of the above factors on the accuracy of the design constraints will be investigated in the next section.

IV. EVALUATION

This section presents the simulations used to verify the proposed approach and investigates its accuracy.

A. Simulations and Setup

To verify the proposal, we compare the calculated values of V_x and g_{om} with HSPICE simulations using AND gates as a case study. Different fan-ins (n_i=1, 3, 4, 5) and fan-outs (n_o=1,

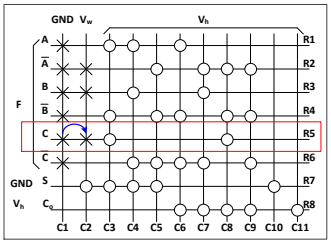


Fig. 5: Example of Sneak Path Currents for a One-Bit Full Adder

3

, 3, 4) are considered. For simplicity, we set V_h to $\frac{V_w}{2}$ [8]. By default, all simulations are performed with Guan's model. The following simulations are conducted.

1) **Default simulation**. The standard Guan's model is used without considering the four factors described in Section III.C.
2) **Impact of device variations**. In these simulations, Guan's model is used to quantify the degree δ_{g0} of the variations; the greater δ_{g0} is, the higher the variation of the memristor devices. Three different values (i.e., δ_{g0}=0.0025, 0.005, 0.01) are investigated.
3) **Impact of the nanowire resistance**. To model the nanowire resistance, a small resistor is added between two memristors in all the rows and columns.
4) **Impact of different memristor models**. Besides Guan's model, we use the ideal [8] and Biolek's (a linear model) [13] model determine V_x analytically, and compare them with Guan's model. Note that the ideal and Biolek's model are simpler models and hence, this simulation shows whether they can be used.
5) **Impact of sneak path currents**. As sneak path currents exist in crossbar, we evaluate their impact using a one-bit full adder based on RBL [9], as shown in Fig. 5. In the figure, '×' at a junction represents a memristor with an unknown logic state and '○' a memristor with high resistance. At the other junctions, no memristors are formed; they are in permanent high resistance R_D (e.g., 10MΩ [15]). The inputs stored in the memristors of column C1 are copied to memristors of column C2 in the same row using one-input AND gates (see row R5 in Fig. 5). For more details about the operation, we refer to [9]. In order to alleviate the sneak path currents, half-select voltages are applied to the rows and columns that are not involved in the operation [8].

All the analytical equations are solved in Matlab, while all simulations are conducted with HSPICE simulator [16]. The memristors and nanowire resistances are described by a SPICE netlist; the memristor models, the CMOS logic that is used to control logic gates, and the voltage drivers are described in Verilog-A. The parameters used in all simulations are summarized in Table II (Guan's model) and II (remaining parameters). The parameters of memristor models and copper nanowires are extracted from [7,8,13,14]. As the switching of output memristors changes the value of V_x, we need to distinguish between different cases; they are 'output memristors are not switching (NSW)', 'before output memristors switch (BSW)' and 'after output memristors switch (ASW)'. Note that V_x is calculated in all three cases, while g_{om} is only calculated after the output memristors switch.

TABLE II: Parameters

Parameter	Description	Value
	General	
F (nm)	Feature size	90
	Ideal and Biolek's Model [8,13]	
R_H (Ω)	High Resistance	200k
R_L (Ω)	Low Resistance	200
R_D (Ω)	Resistance of non-formed memristor	10M
	Nanowire (Copper) [14]	
R (Ω/μm)	Resistance in unit length	9.88
	Control Voltages	
V_w (V)	Program voltage	–
V_h (V)	Half-select voltage	$\frac{V_w}{2}$
	Gate	
n_i	No. of inputs	3,4,5
n_o	No. of inputs	2,3,4
n_L	No. of low-resistance inputs	–

B. Results

The results of all simulations are discussed next.

Default simulation. Table III presents the calculated and simulated results of a AND gate with fan-in=3 and fan-out =2, where V_w=1.4. Both the calculated and simulated g_{om} are normalized by $\frac{g_{om}-g_{min}}{g_{max}-g_{min}}$. Compared to the simulation results, the errors of the calculated V_x and the range of V_w are less than 1%; the error of calculated g_{om} is less than 5%. Therefore, the proposed approach formulates the constraints properly.

Impact of device variation. Fig. 6 shows the calculated and simulated range of V_w for a three-input AND gate with the three different values of δ_{g0}. Fig. 6(a) shows the result for the minimum value of V_w, while Fig. 6(b) the maximum. Compared to the simulation results, the error of the calculated range of V_w is within 3%. In addition, when the variation becomes more serious (i.e., δ_{g0} increases), the error increases only for the minimum calculated V_w.

Impact of nanowire resistance. Fig. 7 shows the calculated and simulated range of V_w for three AND gates (i.e., n_i=3,4,5 and n_o=2). Compared to the simulation results, the error of the predicted range is less than 14%. Note that this error of minimum V_w increases when the number of inputs increases. A gate with more inputs needs a longer row nanowire, and hence the nanowire resistance increases and therefore, also the voltage drop along the nanowire. As a result, the control voltage range for V_w is greater than the calculated values.

Impact of different types of models. Fig. 8 shows the calculated and simulated range of V_w for three AND gates (i.e., n_i=3,4,5 and n_o=2) using different models. The calculated range of V_w based on ideal and Biolek's model are validated with simulations using their corresponding model. For brevity, only the calculated ranges of these three models are shown in Fig. 8. These three models provide a similar minimum V_w around 1.18V. In contrast, the maximum V_w

TABLE III: Verification of the approach

V_x (V)					
n_L	Cases	Calculated	SPICE	Error	Error%
0	NSW	0.6493	0.6500	-0.0007	-0.1077
1	BSW	0.0299	0.0300	-0.0001	0.3333
	ASW	0.2825	0.2810	0.0015	0.5338
2	BSW	0.0159	0.0159	0.0000	0.0000
	ASW	0.2940	0.2920	0.002	0.6849
3	BSW	0.0108	0.0108	0.0000	0.0000
	ASW	0.3007	0.2990	0.0017	0.5686
Normalized g_{om}					
n_L	Cases	Calculated	SPICE	Error	Error%
1	ASW	0.5307	0.5470	-0.0163	-2.9799
2	ASW	0.4273	0.4420	-0.0147	-3.3258
3	ASW	0.3670	0.3810	-0.0140	-3.6745
Range of V_w (V)					
V_w		Calculated	SPICE	Error	Error%
Min		1.1966	1.1968	-0.0002	-0.0167
Max		2.1117	2.1116	0.0001	0.0047

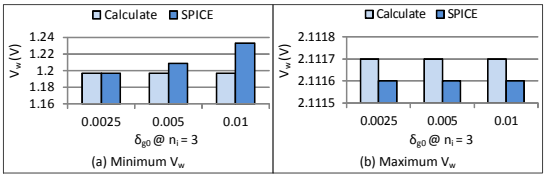


Fig. 6: Impact of Device Variation

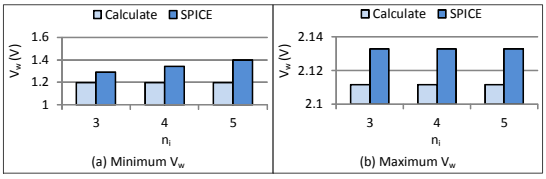


Fig. 7: Impact of Nanowire Resistance

spreads from around 1.5V to 2.1V. Therefore, the way the memristors are modeled impacts the valid range of V_w. Note that the simpler models (e.g., ideal and Biolek) can be used to provide a rough estimation of design constraints, since its valid V_w range is mostly within the range obtained from Guan's model.

Impact of sneak path currents. Table IV shows the calculated and simulated values of V_x, g_{om} and the range of V_w including the impact of the sneak path currents within crossbar. Compared to the simulation results, the error of the calculated V_x and the range of V_w are typically less than 1%; the error of the calculated g_{om} is less than 4%. This is similar to the first simulation which has no sneak path currents. Hence, our approach is able to calculate accurate constraints when sneak path currents exist within a small crossbar.

Above simulations clear show that our proposed approach can effectively calculate the design constraints when the device variation and sneak path currents exist. The errors of the calculated constraints due to nanowire resistance are relatively larger. Future work should therefore also model the nanowire resistance when such constraints are derived. Nevertheless,

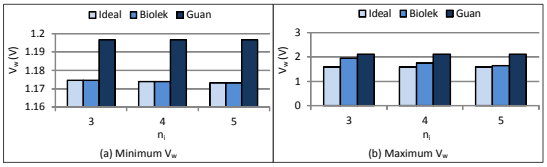


Fig. 8: Impact of Different Models

TABLE IV: Impact of Sneak Path Currents

V_x (V)					
ni0	Cases	Calculated	SPICE	Error	Error%
0	NSW	0.7000	0.6959	0.0040	0.5765
1	BSW	0.0159	0.0179	-0.0020	-11.2049
	ASW	0.2940	0.2923	0.0016	0.5580
Normalized g_{om}					
ni0	Cases	Calculated	SPICE	Error	Error%
1	ASW	0.4273	0.4427	-0.0154	-3.475
Range of V_w (V)					
V_w		Calculated	SPICE	ErrAbs	Err%
Minimum		1.1966	1.1920	0.0046	0.3859
Maximum		2.1117	2.1210	-0.0093	-0.4384

simpler models can provide rough estimations.

V. CONCLUSION

This paper proposed an analytical method to obtain robust memristor-based logic gates. It intensively investigated the impact of device variation, nanowires resistance, sneak path currents, and different types of memristor models. Our method steps closer to the feasible design of memristor logic gates.

REFERENCES

[1] ITRS, "Beyond cmos white paper," 2014.
[2] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *DATE*. IEEE, 2017.
[3] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015, pp. 1718–1725.
[4] X. Zhu *et al.*, "Performing stateful logic on memristor memory," *TCAS–II*, 2013.
[5] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *DTIS*. IEEE, 2016.
[6] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *TVLSI*, 2014.
[7] X. Guan *et al.*, "A spice compact model of metal oxide resistive switching memory with variations," *EDL*, 2012.
[8] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *ICCD*, 2015.
[9] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, 2005.
[10] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, 2009.
[11] C. Yakopcic *et al.*, "Memristor spice modeling," in *Advances in Neuromorphic Memristor Science and Applications*.
[12] E. Linn *et al.*, "Applicability of well-established memristive models for simulations of resistive switching devices," *TCAS-1*, 2014.
[13] Z. Biolek *et al.*, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, 2009.
[14] G. S. Snider *et al.*, "Nano/cmos architectures using a field-programmable nanowire interconnect," *Nanotechnology*, 2007.
[15] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Adv. Mat.*, 2011.
[16] Synopsys hspice user guide.

4

INTERCONNECT DESIGN

This chapter presents the design of interconnects which are used to connect primitive logic gates. First, it explores and compares three possible schemes to implement interconnect networks: a) using only the memristor crossbar, b) using only the CMOS peripheral circuits, c) using both the memristor crossbar and CMOS pass transistors. Then, it presents interconnect schemes for both intra-tile and inter-tile communication, where a tile is a building block or processing element.

The content of this chapter consists of the following research articles:

1. *H.A. Du Nguyen, **L. Xie**, M. Taouil, S. Hamdioui, K.L.M. Bertels, Interconnect Networks for Resistive Computing Architectures, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Dresden, Palma de Mallorca, Spain, 2017, pp. 1-6*
2. ***L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, Interconnect Networks for Memristor Crossbar, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, July, 2015, pp. 124-129*

4.1. INTRODUCTION

Interconnects are required to connect primitive logic gates employed by a complex circuit. Therefore, finding the best method to implement the interconnects is an important question. Another interesting question is further exploring the use of crossbar-based interconnect to support both intra- and inter-tile communication. This chapter will present these aspects.

Exploration of Interconnect Implementations: Although there is some research focusing on implementing individual logic gates, research on implementing the interconnect network used to connect individual gates is not explored at all. Therefore, it is of great interest to explore different possible implementation methods of interconnect networks for large scale circuits.

Interconnects for Intra-tile and Inter-tile Communication: The fundamental function of interconnects is supporting the communication within a tile (i.e., a building block or IP) or between tiles. However, how to use crossbar-based interconnect to support such intra-tile and inter-tile communication has not been studied yet.

4.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

- *Exploration of Interconnect Implementations* [54]: Three methods are explored and compared to implement interconnect networks used in logic circuits. First, we only use the memristor crossbar to build the interconnect, such type of interconnect uses copy operations within crossbar to transfer data. The second method is using the CMOS peripheral circuit of memristor arrays. The peripheral circuit first reads out the data from the source and then writes the data to the destination. The final method is using some pass transistors to directly connect the source and destination. To evaluate and compare these three methods, three parallel adders are implemented using different methods. The SPICE simulation results show that the third method speeds up the data transfer based on the first two. My work is focusing on the interconnects using only memristor crossbars.

- *Crossbar Based Interconnects for Intra-tile and Inter-tile Communication* [55]: An intra-tile interconnect for generic logic functions is first proposed; it transfers data between gates or building blocks based on copy operations executed within crossbars. Such an interconnect network can be used route signals for ASICs. In addition, we design a dedicated intra-tile interconnect network to transpose a matrix. It consists of an input, intermediate and output blocks. The input and output blocks store the original and transposed matrix, respectively. The intermediate block stores one column of the input matrix at a time and the, the transposed column is transferred to the output block. This is repeated for all the input columns. To implement inter-tile communication, we propose 2D bus, which enables both horizontal and vertical transmission between tiles. A 2D bus consists of some buffers implemented by reserved memristive devices within the crossbar.

These buffers allow the 2D bus and tiles to run without blocking each other. The 2D bus is able to support three basic communication patterns including unicast, multicast, and broadcast. These interconnect networks are validated with SPICE simulations.

Interconnect Networks for Resistive Computing Architectures

H.A. Du Nguyen, Lei Xie, Jintao Yu, Mottaqiallah Taouil, Said Hamdioui
Laboratory of Computer Engineering, Delft University of Technology, The Netherlands
Email: H.A.DuNguyen@tudelft.nl

***Abstract*—Today's computing systems suffer from a memory/communication bottleneck, resulting in high energy consumption and saturated performance. This makes them inefficient in solving data-intensive applications at reasonable cost. Computation-In-Memory (CIM) architecture, based on the integration of storage and computation in the same physical location using non-volatile memristor crossbar technology, offers a potential solution to the memory bottleneck. An efficient interconnect network is essential to maximize CIM's architectural performance. This paper presents three interconnect network schemes for CIM architecture; these are (1) CMOS-based, (2) memristor-based and (3) hybrid cmos/memristor interconnect network scheme. To illustrate the feasibility of such schemes, a CIM parallel adder is used as a case study. The results show that the hybrid interconnect network scheme achieves a higher performance in comparison with the CMOS-based and memristor-based interconnect scheme in terms of delay, energy and area.**

4

I. INTRODUCTION

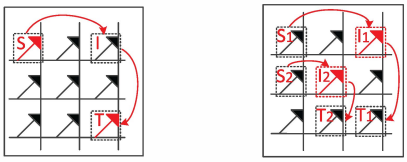
Classical Von Neumann computer architectures are unable to deal with big data problems efficiently due to the memory bottleneck [1], high energy consumption [2] and inefficient programming methodology [3]. Architectures based on resistive computing [4–6], such as Computation-in-Memory (CIM) architecture [6], are emerging and try to address the aforementioned problems [7]. CIM architecture alleviates the memory bottleneck by using the same physical devices to implement both logic and storage units. However, such computing architectures require efficient interconnect network scheme to explore their potential. The interconnect network and communication schemes applied in CMOS technology are not applicable to memristor crossbar due to fundamental differences in their working principle. In CMOS technology, logic signals are represented by voltages that propagate through wires. In CIM, logic signals are represented by resistances that are programmed by control voltages. Therefore, it requires a distinct interconnect network.

Recent research in the field has been focusing on implementing logic inside the memristor crossbar [8–11]. However, limited work has investigated possible interconnect network and communication schemes; so far only a single publication has addressed this topic. In previous work [12], the authors have proposed different communication schemes (i.e., unicast, multicast, and broadcast) using an interconnect network fully integrated in the memristor crossbar. The scheme has a high area overhead and the communication cost depends on the relative positions between the source and target memristor. This complicates the place-and-route phases [13]. Therefore, there is a need to explore different possible interconnect network schemes in order to optimize overall performance.

In this paper, we develop and investigate the potential of different interconnect network schemes and use the CIM parallel adder [14] as a case study. Note that the proposed interconnect network schemes can be applied to any crossbar-based resistive computing architecture. The contributions of this paper are the following:

- We propose two new interconnect networks for CIM architecture.
- We compare them with the previously interconnect network proposed in [12]; the CIM parallel adder is used as a case study.
- We evaluate the overhead of the three interconnect network schemes in terms of delay, energy and area.

The rest of this paper is organized as follows. Section II briefly describes the state-of-the-art and the CIM parallel adder implementation. Section III presents the interconnect network schemes. Section IV discusses the evaluation model and results. Finally, Section V concludes this paper.



(a) Single *copy* operation (b) Diagonal scheme

Fig. 1: Interconnect Network Proposed in [12]

II. BACKGROUND

This section first presents the state-of-the-art in memristor crossbar based interconnect networks. Thereafter, it discusses the CIM parallel adder and its crossbar implementation [10].

A. State-of-the-Art Interconnect Network Schemes

In [12], the authors proposed a complete interconnect network for the memristor crossbar. Copy operations are used to move data between two memristors. A *copy* operation is carried out by applying appropriate control voltages to the source and target memristor nanowires. A direct *copy* operation takes one memristor write cycle and occurs when both memristors share the same horizontal or vertical nanowire.

To perform a *copy* operation between two memristors residing on different rows and columns, an extra copy operation is required as shown in Fig. 1a. Here, an additional memristor located at one of its two intersection points referred to as diagonal scheme. In the figure, S presents the source memristor and T the target memristor, while I is used as intermediate memristor. As a result, the communication cost doubles, i.e., two memristor write cycles. In order to transfer multiple bits simultaneously, the diagonal scheme can be extended (as shown in Fig. 1b); it performs the data transfers (i.e., from S_1 to T_1 and S_2 to T_2) in two cycles.

978-1-5090-6377-2/17/$31.00 ©2017 IEEE

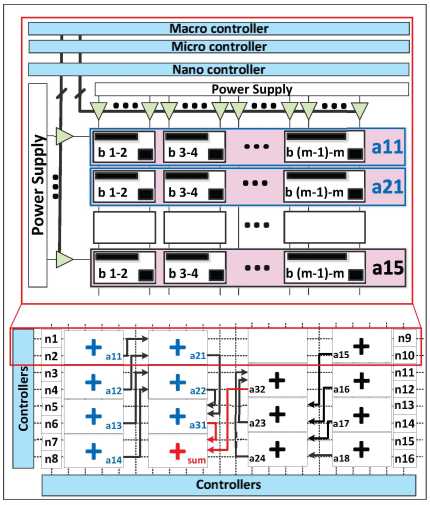


Fig. 2: CIM-based Parallel Adder (16 inputs)

In [8], the authors present a memristor adder based on Complementary Resistive Switches (CRS). As both high and low resistance have the same equivalent resistance, copy operations cannot be used. Instead, the authors propose a CMOS interconnect network consisting of a controller with additional logic (such as sense amplifier, write driver) to read from the source memristor and write to the destination. However, no implementation details were reported.

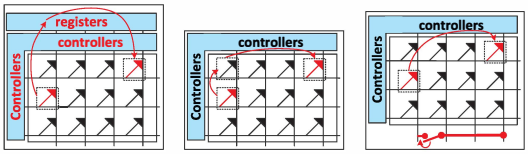
B. CIM Parallel Adder

The lower part of Fig. 2 shows a parallel adder (based on the binary tree) implemented using the CIM concept, referred to as CIM parallel adder [14]. It consists of two components: (i) the computation and storage part which reside in the memristor crossbar, and (ii) the CMOS controlling part.

The memristor crossbar has two purposes, i.e., it performs computations (additions) and it is used as a storage (memory) device. In Fig. 2, the adders are indexed by a_{xy} where x presents the addition stage and y the adder index within a stage. The adders in this architecture are not reused as they relatively do not impact the interconnect network schemes. Dedicated storage cells for the inputs are presented by n_i, where i the index of the i^{th} input. Each storage cell consists of a memristor that is able to store a single bit. The placement of the adders and memory is flexible as both are implemented using memristors. To utilize the area of the binary tree implementation efficiently, half of the inputs are mapped from left side, while the other half from right side.

The upper part of Fig. 2 shows the CIM parallel adder implementation using the Fast Boolean Logic Circuit (FBLC) style [10]. Each adder a_{xy} is replaced by an FBLC 32-bit ripple carry adder that is composed of 16 two-bit adders; the two-bit adders are selected for their area efficiency [10]. Each two-bit adder is represented by a rectangle with two dark areas in the upper part of Fig. 2; the longer dark areas present the input area, while the smaller dark areas the outputs. The crossbar, the controller, and the peripheral circuit are marked in pink, blue and green area, respectively.

Fig. 3: State Machine for Micro Controller



(a) CMOS (b) Memristor (c) Hybrid

Fig. 4: Proposed Interconnect Network

The CMOS part consists of the controller and peripheral circuit. The controller is hierarchically divided in three layers, i.e., macro, micro and nano controller. The macro-controller controls the addition stage of the binary tree. The micro-controller executes a single addition. Each two-bit addition has four states as shown in Fig. 3; these states are repeated 16 times to complete a 32-bit addition. The nano-controller translates the macro- and micro-controller instructions to appropriate control voltages which are subsequently applied to the appropriate nanowires of the crossbar. Each controller is implemented with a state machine. More details of the nano-controller can be found in [10,12].

The crossbar implementation requires a peripheral circuit to support the memristor crossbar operations. For example, the peripheral circuit includes voltage drivers to control the nanowires, sense amplifiers to read out memristor states, and multiplexers accompanied with the sense amplifiers to select which nanowires are accessed, and logic circuitry to create connections between different crossbar nanowires.

III. INTERCONNECT NETWORK SCHEMES

This section describes the overview of interconnect network schemes. Thereafter, it integrates them in the CIM parallel adder. Note that the CIM parallel adder is only used for illustrative purposes. The proposed interconnect networks can be applied in general to any resistive computing architectures.

A. Overview

Fig. 4 shows the three proposed interconnect networks. They are referred to as: (1) CMOS, (2) memristor, and (3) hybrid scheme. Each scheme impacts the crossbar and CMOS logic differently, as explained next.

Fig. 4a shows the CMOS interconnect network scheme. In this scheme, data movements in the crossbar transit through registers in the CMOS layer and consist of two memristor operations (i.e., a read and write operation). First, the controller reads the value from a source memristor. It stores the value temporarily in a CMOS register. Second, it writes the register value into the target memristor. This interconnect network scheme requires sense amplifiers, multiplexers, registers and a controller that applies appropriate control voltages. The communication requires two cycles with a the cycle time equal to the maximum delay of the read and write operation. The delay of the read operation consists of the delay of the controller, sense amplifiers, multiplexers and the time to write the CMOS registers. The write path delay consists of the controller delay and the write time of the target memristor.

Fig. 4b shows the memristor interconnect network scheme. In this scheme (based on the work published in [12]), the

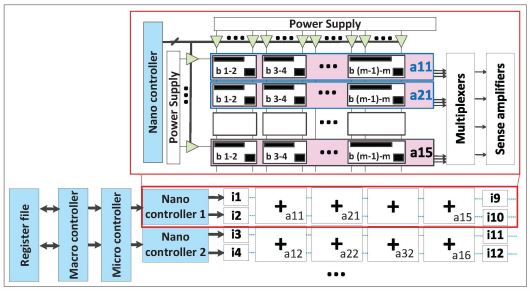


Fig. 5: CMOS-based Interconnect Network

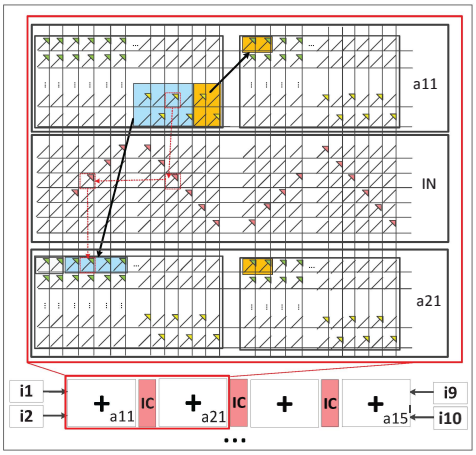


Fig. 6: Memristor-based Interconnect Network

controllers need one or two steps to transfer data from the source to target memristor. Each step consist of a memristor *copy* operation. This scheme requires an extra intermediate memristor at the intersection points between the source and target memristors. In case multiple bits are transferred simultaneously, the diagonal scheme may be used [12]. The maximum delay of the scheme is two memristor write cycles in case there are no conflicts between intermediate and reserved (for logic or storage) memristors. In case of a conflict, more than two copy operations are required.

Fig. 4c shows the hybrid interconnect network scheme. This scheme also uses a *copy* operation to transfer data from source to target memristor. It performs this operation in a single cycle by creating a direct path between source and destination using a CMOS switch as depicted in the bottom part of Fig. 4c. This switch can be implemented by a pass-gate. The transfer delay equals the delay of the controller to perform the *copy* operation and to activate the pass transistor.

As described above, the three proposed schemes are capable of transferring data between any two memristors, and support parallel communication. However, they impact the delay, energy and area consumption of the memristor crossbar and CMOS control differently. Note that the above concept applies to data transfers between adders within the same sub-crossbar (intra-crossbar) and two different sub-crossbars (inter-crossbar). In the following sections, the three schemes are applied to the CIM parallel adder as a case study.

B. CMOS Interconnect Network

Fig. 5 shows the CIM parallel adder based on the CMOS interconnect network scheme. Each sub-crossbar has its own nano-controller. Therefore, communication within a sub-crossbar is handled by a single controller while two controllers are involved in the communication between different crossbars.

In intra-crossbar communication, (e.g., between adders $a11$ and $a21$), the controller reads out the value from the source adder ($a11$), stores it in a register, and writes the value in the target adder ($a21$). In inter-crossbar communication, (e.g., between adders $a12$ and $a21$), the communication is handled by two controllers and may be implemented in two ways. The first method adds an additional network on the CMOS side to move data between the controllers. For example, a source controller (e.g., *nano-controller 2* in Fig. 5) reads out the value of adder $a12$, transmits the value via the interconnect network (which is also implemented by CMOS devices) to the target controller *nano-controller 1*; the target controller writes the received values into the inputs of the target memristors (i.e., inputs of $a21$). The second method is through a register file. In this case, the source controller writes the read-out value to a predefined register; thereafter, the target controller reads the value from the predefined register. Note that, the register file may become large when there are a lot of data transfers. In case the size is too big, a memory can be used.

A read-out operation might destroy the value stored in the source adder [8]. Hence, a write-back (restore) operation is also required. The read-out, write-in and write back operations are carried out by applying appropriate control voltages to the nanowires and are required for each data transfer.

Due to the synchronization between two controllers, the read-out and write-in operations executed sequentially using two different steps, while the write-back operation is executed simultaneously with the write-in operation. Therefore, with respect to the crossbar, the delay equals two memristor write operations, the energy three memristor write operations per bit (including the write-in, write-back operation and the possible destructive read of the source memristor), and the area is zero (as no extra memristors are required). With respect to the controller, each crossbar needs its own nano-controller. In addition, two states are required for the two communication steps in the micro controller in comparison with the original design [12]. With respect to the peripheral circuit, a 2-bit adder has three values to be read out (one carry bit and two sum bits) simultaneously; hence three sense amplifiers are required per adder. As only one 2-bit adder is active at a time in a sub-crossbar, the sense amplifiers can be shared, while multiplexers are used to forward the data from the currently working adders. In short, only three sense amplifiers are required per sub-crossbar. Note that the sense amplifier may be a current [15] or voltage sense amplifier [16] depending on the adder logic type. In the CIM parallel adder implementation using FBLC style, a voltage sense amplifier is preferred [16]. Finally, a register file is required to store temporary read values.

C. Memristor Interconnect Network

Fig. 6 shows the CIM parallel adder based on the memristor-based interconnect network scheme. In this scheme, we use the state-of-the-art interconnect network proposed in [12]. As the

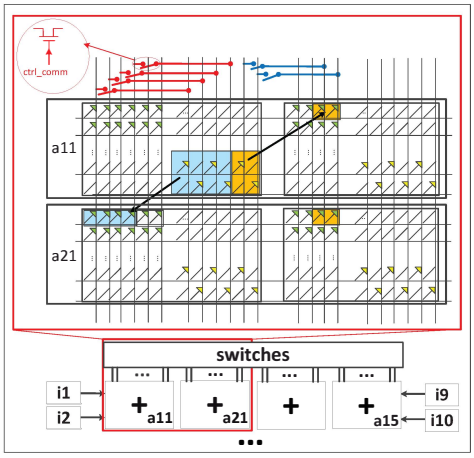


Fig. 7: Hybrid Interconnect Network

FBLC adder is used, the nano-controller can be shared among the sub-crossbars as they require the same control voltages at each time step.

Note that each adder is built from multiple two-bit adders that are placed on the same adder row. To simplify the state machines, communication takes directly place after partial results are obtained. Consider for example the output of adder $a11$ which will be transferred to the input of adder $a21$. The target adder $a21$ is located in a different row. As soon as the first two-bit adder of adder $a11$ produces its outputs, they will be moved to the proper locations. More precisely, the carry output (in orange in Fig. 6) of this adder needs to be transferred to the second two-bit adder of $a11$, while the sum output (in blue in Fig 6) is needed at the first two-bit adder of $a21$ (for the addition in the next stage). As the source and target memristors are on different rows, additional intermediate memristors (see Fig. 1) are required to create a path between them. The red dotted arrow lines present the communication paths between source and destination adder.

As multiple bits need to be transferred simultaneously, the diagonal scheme of Fig. 1b is used as shown in the upper part of Fig. 6; denoted by Interconnect Network (IN). In this scheme, all communications from one adder stage to the next one can be performed simultaneously with a latency of three cycles due to two intermediate hops. Note that only one IN block is needed per sub-crossbar as one adder is active at the time.

Next, we discuss the delay and area cost. With respect to the crossbar, the communication delay between source and destination equals 3 write operations for each n=2-bit addition, the energy equals 3 memristor write operations per data transfer (as 3 writes are required due to the need of two intermediate memristors). Each sub-crossbar needs $2 \cdot (n+1)$ additional rows for the IN block as an n-bit adder has n-sum bits and one carry bit and their complementary values as outputs. With respect to the CMOS part, three additional states are required for the micro-controller to perform the communication steps. With respect to the peripheral circuit, six additional voltage drivers are required per sub-crossbar to drive the rows of the IN block.

TABLE I: Simulation Parameters [9,18]

Parameters	Value	Parameters	Value	Parameters	Value
R_L	$100k\Omega$	R_H	$1G\Omega$		
V_w	1.4V	V_{hw}	0.7V	V_{th}	1V

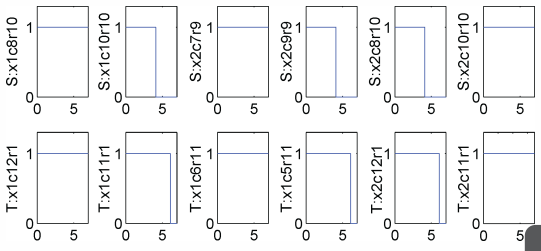


Fig. 8: HSPICE verification of hybrid communication schem

4

D. Hybrid Interconnect Network

Fig. 7 shows the CIM parallel adder based on the hybrid interconnect network scheme which consists of both CMOS devices and memristors. This scheme also utilizes the memristor *copy* operation as mentioned in [12]. However, instead of using additional intermediate memristors, CMOS switches are used to create direct paths between source and target memristors. For instance, in the upper part of Fig. 7 two nanowires are connected by a transmission-gate [17], which is controlled by the CMOS controller. When the switch is open, the two nanowires are disconnected and normal computations can be performed. As the switch closes, the memristors on the two different nanowires are directly connected. Therefore, a single *copy* operation [12] can be used to transfer data within and between sub-crossbars. As the inputs of the target adder and the outputs of the source adder are both effectively located in the same row, multiple bits can be copied simultaneously.

We discuss the cost of the scheme next. With respect to the crossbar, the delay of a *copy* operation equals one write operation, the energy consists of writing the target memristor (per single *copy* operation [12]), and the area is zero (as no extra memristors are required). With respect to the CMOS part, additional logic is required to control and implement the switches [17]. The micro controller requires one additional state to perform the communication step. No sense amplifiers and multiplexers are required for the peripheral circuit.

IV. RESULTS

This section first verifies the proposed interconnect network schemes. Thereafter, we present a model to evaluate them. Last, we discuss the performance results.

A. HSPICE Verification

We verify the proposed interconnect network schemes in HSPICE simulations using the ideal memristor model [9,18] using the same simulation approach as in [12]. The memristor parameters are shown in Table I; the low resistance R_L = $100k\Omega$ represents a logic 0, the high resistance R_H = $1G\Omega$ represents a logic 1, the memristor threshold voltage V_{th} equals 1V, the memristor write voltage V_w 1.4V, and the half-select voltage V_{hw} 0.7V. Due to space limitations, only the results of the hybrid scheme are included. Fig. 8 shows the data transfer

TABLE II: Basic Model Parameters

Component	Basic Unit	Parameter	CMOS-based	Memristor-based	Hybrid
Device Technology					
Memristor Technology		F_m - Feature size (nm)	5 [19]		
		D_m - Delay (ps)	200 [19]		
		E_m - Energy (fJ)	1 [19]		
		A_m - Area (nm^2)	100 ($4F_m^2$) [19]		
CMOS Technology		F_c - Feature size (nm)	40		
CIM Parallel Adder Interconnect Network Schemes					
Peripheral Circuit	Voltage driver	D_{icc} - Delay	not applicable	masked by adder voltage driver	
		E_{vd} - Energy (pJ)		E_{pt}	E_{pt}
		A_{vd} - Area (um^2)		$3 \cdot A_{pt}$	A_{pt}
	Sense Amplifier [16]	D_{sa} - Delay (ps)	20	not applicable	
		E_{sa} - Energy (pJ)	3.69		
		A_{sa} - Area (um^2)	0.017		
	Multiplexer [20]	D_{mux} - Delay (ps)	D_{pt}		
		E_{mux} - Energy (pJ)	E_{pt}		
		A_{mux} - Area (um^2)	$16 \cdot 4 \cdot A_{pt}$		
Controller (Synthesized for 16 inputs, 32-bit input)		D_{cont} - Delay (ps)	825	825	825
		E_{cont} - Energy (pJ)	0.115	0.103	0.100
		A_{cont} - Area (um^2)	2234	1869	1147
Memristor Crossbar		D_{icx} - Delay (ps)	400 (2 cycles $\cdot$ D_m)	600 (3 cycles $\cdot$ D_m)	200 (1 cycle $\cdot$ D_m)
		E_{icx} - Energy (fJ)	2 (2 memristors $\cdot$ E_m)	3 (3 memristors $\cdot$ E_m)	1 (1 memristors $\cdot$ E_m)
		A_{icx} - Area (um^2)	0	200 (2 memristors $\cdot$ A_m)	0

from source memristors (first row) to the target memristor (second row); in the y-axis, S and T represent the source and target memristor, x the crossbar index, c and r the column and row index; the x-axis shows the time steps. The data values are selected randomly. The first two plots of both the source and target memristors present simultaneous *copy* operation that occurs in one single sub-crossbar from different source rows ($r10$ and $r9$) to one target row ($r1$) (see orange blocks in Fig. 7). The next two plots present similar *copy* operations which take place between two different sub-crossbars (from the output rows of sub-crossbar $x1$ to the input row of sub-crossbar $x2$, e.g., as shown by the blue blocks in Fig. 7). The last four plots shows that simultaneous *copy* operations can occur within the same or between different sub-crossbars.

B. Evaluation Model

The interconnect network schemes are evaluated using the CIM parallel adder model. The model includes four components: memristor adders, peripheral circuit (voltage drivers, sense amplifiers, multiplexers), controller (including register file). The basic adder implementation is the same for the three interconnect schemes. Table II shows the basic parameters of the model. The top part of the table shows the memristor and CMOS technology parameters. The memristor device has a feature size F_m of $5nm$, delay D_m of $200ps$, energy consumption E_m of $1fJ$ per transition, and area A_m of $100nm^2$. The CMOS device has a feature size F_c of $40nm$. This technology node is also used to synthesize the controller. The lower part of Table II shows the basic parameters of the peripheral circuits, controller.

The above basic model parameters are used to estimated the performance of the proposed interconnect network schemes. The delay per cycle is calculated by the sum of the components that are part of the critical path. The total delay is calculated by taking the product of the delay per cycle and the number of cycles. The total energy and area are obtained by summing up the energy and area cost of all the sub-components.

C. Results

Fig. 9 shows the total delay, energy and area of the CIM parallel adder implementation with the three proposed interconnect network schemes, while Fig. 10 shows the cost breakdown per component. Fig. 10 contains six plots related to the energy and area of each component (i.e., crossbar, controller and peripheral circuit). The delay is not considered here due to space limitations; in addition, the delay is based on the critical path and not on the total sum of the individual components. The results of each metric are described next.

Delay: In terms of delay, the hybrid scheme outperforms marginally the other two schemes as it (i) does not have a sense amplifier and multiplexer delay in its critical path, (ii) requires fewer controller states, and (iii) needs fewer memristor writes operations for communication.

Energy: In terms of energy consumption, the memristor-based scheme performs the worst due to the additional writes required for the intermediate memristor and control states. In comparison with the CMOS-based scheme, the hybrid scheme has a lower energy, as it requires fewer CMOS devices and has a lower delay.

The energy breakdown for each component is shown in the top part of Fig. 10. Note that the interconnect network is only a small part of the FBLC based CIM parallel adder, and therefore, nearly the same energy results are observed for the crossbar part. Nevertheless, the memristor-based scheme has the highest crossbar energy consumption due to the writing of the intermediate memristors. In comparison with the CMOS-based scheme, the hybrid scheme has slightly a lower energy consumption as each data transfer requires a single write. With respect to the controller, the CMOS-based scheme has the highest energy consumption due to the additional control logic and register file. In the CMOS-based scheme, read-out operations may be destructive and write-back operations are required to preserve the value of the source memristor, hence the additional states also cost more energy. Note that the hybrid scheme's controller requires only one additional state for the micro-controller, while the CMOS- and memrisor- based schemes need two and three additional states, respectively. With respect to the peripheral circuit, the CMOS-based scheme also has the highest energy consumption, due to the presence

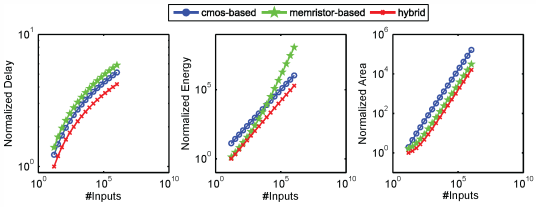


Fig. 9: Performance of Proposed Interconnect Network Schemes

of sense amplifiers and multiplexers. Note that the switches used for the interconnect network in the hybrid scheme show relatively a marginal energy consumption. The memristor-based scheme has a slightly higher energy consumption in comparison with the hybrid scheme, mostly due to the presence of additional voltage drivers.

Area: In terms of area, the CMOS-based scheme has the highest area overhead due to the presence of additional peripheral circuits, a register file and a larger controller. The other two schemes have nearly the same area overhead. We will discuss the cost breakdown next.

Each component's area is shown in the lower part of Fig. 10. With respect to the crossbar, the memristor-based scheme consumes slightly more area than the other two schemes as additional memristors are required to perform the communication. With respect to the controller, the hybrid scheme has the smallest area overhead as only one additional state is required for the micro-controller. The controller of the memristor-based scheme has slightly a smaller area than the CMOS-based scheme, as the CMOS-based scheme needs a register file and additional logic to address the register file. With respect to the peripheral circuit, the memristor-based scheme has slightly a higher area overhead as compared to the other two schemes due to the presence of voltage drivers. The peripheral circuits of the hybrid scheme requires less area than the CMOS-based scheme due to the presence of sense amplifiers and multiplexers.

In summary, the hybrid communication scheme shows benefits in terms of delay, energy and area due to fewer control states, less peripheral circuits and no need of additional memristors are required for the interconnect network. The results of this paper are useful for memristor designers, as it is essential to select appropriate interconnect schemes to improve the overall performance.

V. CONCLUSION

In this paper, we showed three feasible interconnect network schemes for resistive computing architectures in general, and for CIM architecture in particular. The result showed that a hybrid interconnect network scheme has the highest efficiency in terms of delay, energy and area. This shows that an interconnect network fully integrated in the memristor crossbar is not efficient enough, and can lead to vast performance reduction. Similarly, an interconnection network solely based on CMOS devices also showed to be inefficient. In conclusion, it is essential to utilize both memristor operations and CMOS devices to build an effective interconnect network scheme for resistive computing architectures.

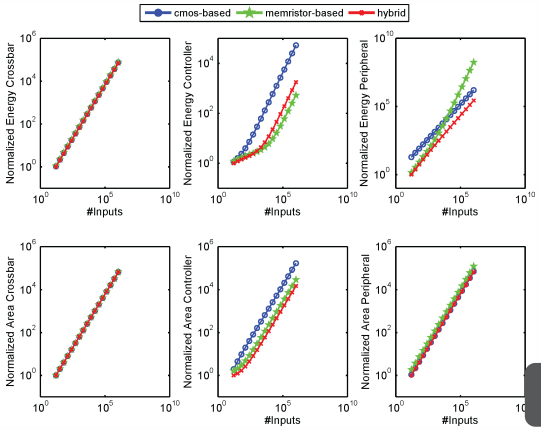


Fig. 10: Performance Breakdown of Each Component

REFERENCES

[1] S. Kaxiras, *Architecture at the End of Moore*, ser. Advances in Atom and Single Molecule Machines. Springer Berlin Heidelberg, 2013.
[2] B. Hoefflinger, "The energy crisis," in *Chips 2020*, ser. The Frontiers Collection, 2012, pp. 421–427.
[3] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, pp. 365–376, 2011.
[4] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (plim) computer," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.
[5] R. B. Hur *et al.*, "Algorithmic considerations in memristive memory processing units (mpu)," *International Symposium on Nanoscale Architecture (NANOARCH)*, 2016.
[6] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *IEEE Conference on Design, Automation & Test in Europe*, 2015, pp. 1718–1725.
[7] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Design, Automation and Test in Europe DATE*, 2017.
[8] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, pp. 64–74, 2015.
[9] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.
[10] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *IEEE Conference on Computer Design (ICCD)*, 2015, pp. 335–342.
[11] S. Kvatinsky *et al.*, "Magicmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.
[12] L. Xie *et al.*, "Interconnect networks for memristor crossbar," in *Symposium on Nanoscale Architectures*, 2015, pp. 124–129.
[13] J. Yu *et al.*, "Skeleton-based design and simulation flow for computation-in-memory architectures," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*. IEEE, 2016, pp. 165–170.
[14] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *Symposium on Nanoscale Architectures*, 2015, pp. 57–62.
[15] S. Sundaram *et al.*, "High speed robust current sense amplifier for nanoscale memories: a winner take all approach," in *Conference on VLSI Design (VLSID)*, 2006.
[16] W. Dehaene *et al.*, "Variability-aware design of low power sram memories," *Doctoral Thesis*, 2009.
[17] R. Zimmermann *et al.*, "Low-power logic styles: Cmos versus pass-transistor logic," *IEEE journal of solid-state circuits*, vol. 32, 1997.
[18] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.
[19] ITRS, "The international technology roadmap for semiconductors ITRS," Semiconductor Industry Asso, Tech. Rep., 2011.
[20] SYNOPSYS, "Digital standard cell library saed_edk90_core databook," SYNOPSYS ARMENIA Educational Department, Tech. Rep., 2008.

Interconnect Networks for Memristor Crossbar

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering
Delft University of Technology, Delft, The Netherlands
Email: L.Xie@tudelft.nl

***Abstract*—As the down-scaling of CMOS technology reaches inherent physical device limits, major challenges arise such as reliability, power consumption, etc. Novel technologies are under investigation as an alternative for next-generation VLSI circuits. Memristor is one of the promising candidates due to its scalability, non-volatility, practically zero leakage, high integration density, etc. Several applications have been proposed for the memristor crossbar architecture; examples are neuromophic, memories and logic circuits. This paper proposes a generic and flexible interconnect network for memristor-based logic circuits within a crossbar architecture. The scheme is based on the use of reserved memristors (within the crossbar) and the execution of copy operations from and/or to the reserved memristors; it can be applied both at intra-tile as well as inter-tile level, where a tile is a basic block performing a simple logic function. To illustrate the potential of the proposed scheme, two intra-tile and one inter-tile case study are evaluated using SPICE simulation, and their incurred costs are analyzed; the case studies consist of interconnect networks for (a) a specific function within a tile (i.e., the matrix transpose), (b) generic logic functions within a tile, and (c) communication between tiles.**

I. INTRODUCTION

The down-scaling of CMOS technology gradually reaches the inherent physical device limits and leads to significant challenges [1]; e.g., saturated performance gain, higher leakage, reduced reliability and complex fabrication process. To deal with these issues, emerging technologies (e.g., graphene transistors, nanotube, tunnel field-effect transistor, memristors, etc. [2]) are proposed as an alternative for next-generation VLSI circuits. Among the emerging technologies, memristor is a promising candidate. It provides great scalability, high density, zero standby power consumption, higher technology maturity, CMOS process compatibility, and the ability to implement both logic and memory in the same physical crossbar [3].

Memristors can be utilized as individual discrete devices or in arrays [4]. Designs using discrete memristors are intended to improve the performance of conventional circuits [4] such as chaos circuits, analog circuits (e.g., variable gain amplifier), etc. Memristor arrays provide numerous memristors which fit straightforward on a crossbar architecture [5]. Here, memristors are located at the intersection of horizontal and vertical nanowires. Therefore, a crossbar architecture can provide a high integration density, and therefore enable many applications such as neuromorphic systems [5], non-volatile memories [4], new computing paradigms for data-intensive applications [6], etc.

Research on memristor-based logic circuits has started getting more attention since the fabrication of the memristor device by HP in 2008. Four types of such circuits have been proposed: (i) Boolean logic [7], (ii) material implication logic [3], (iii) threshold logic [8], and (iv) majority logic [8]. Although memristor circuit design is becoming more popular, the research on interconnects for such circuits is still in its infancy stage. For circuits based on discrete memristors, metal wires for the implementation of the interconnections -similarly as in the conventional CMOS logic circuits- were proposed in [8], while for crossbar arrays, the author [7] proposed the concept of using circuit switching (no simulations were included); circuit switching uses nanowires and memristors as routing switches. Obviously, using traditional metal wires for dense crossbar will induce serious delay and significantly increase the routing complexity. On the other hand, circuit switching requires the programming of memristors to form a low-resistance path between different parts of the circuit prior to transmitting data; creating such low-resistance paths might be infeasible due to sneak path currents.

This paper proposes a generic and flexible interconnect network using primitive low-cost copy operations for crossbar architectures; the scheme is resilient against sneak path currents. This interconnect network can be used both at intra- and inter-tile levels. The contributions of this paper are:

- A methodology to design intra- and inter-tile interconnect networks.
- An interconnect network supporting several communication schemes such as unicast, multicast and broadcast.
- A diagonal data storage format reducing the communication latency.
- Different interconnect networks for three case studies: (a) a specific function within a tile (i.e., the matrix transpose), (b) generic logic functions within a tile, and (c) communication between tiles.

The remainder of this paper is organized as follows. Section II briefly describes some background regarding the memristor model. Section III discusses the fundamentals of the proposed interconnect network and its applications. Section IV presents intra- and inter-tile interconnect networks as case studies. Section V presents the simulation results. Finally, Section VI concludes the paper.

II. BACKGROUND

This section describes the memristor model, its data representation and control voltages.

978-1-4673-7849-9/15/$31.00 ©2015 IEEE

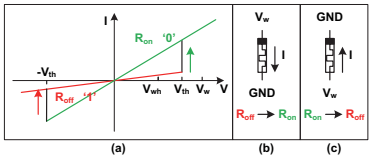


Fig. 1: A simplified memristor THAB model [3].

A. Memristor Model

Although memristors have different physical mechanisms [9], their behaviour at circuit level can be abstracted by two parameters: resistive switching behaviour (abrupt or smooth) [10] and switching threshold voltage (with or without a minimum voltage for resistive switching) [11]. In this work we use a simplified model with abrupt switching and switching threshold voltage [3,7] (see Fig. 1(a)), referred to as THAB (THreshold-ABrupt) memristor model. The model is suitable for digital circuits and avoids issues such as memristance drift and undesired switching [12].

Fig. 1(a) shows the characteristic of THAB model; the black square in the figure presents the positive terminal of the memristor. The memristor switches from one resistive state to another when the absolute value of the voltage (either positive or negative) across the device is greater than its threshold voltage. Otherwise, it stays in its current resistive state. Typically, a memristor requires two different switching threshold voltages to switch from high to low resistance and vice versa [11]. For simplicity, we assume that THAB model uses the same absolute threshold voltage V_{th} for both cases.

B. Data Representation and Control Voltages

A memristor has two resistive states, high R_{off} and low R_{on} resistance (see Fig. 1(a)). In this paper, R_{off} and R_{on} present a logic 1 and 0, respectively. To control the digital circuits, three different voltages are required: V_w, V_{wh}, and *GND*. V_w is used to program the resistance of a memristor as shown in Fig. 1(b) and (c), and V_{wh} is used to alleviate the impact of sneak path currents by using half-select voltage strategy [5]. V_{wh} is applied to memristors which are not accessed. The relationship between V_w, V_{wh}, *GND* and V_{th} is [5]: $0<V_{wh}=\frac{V_w}{2}<V_{th}<V_w$. This guarantees $V_w-V_{wh}=2V_{wh}-V_{wh}=V_{wh}<V_{th}$, preventing undesired switching.

III. Fundamentals of Interconnect Network

This section describes briefly the requirements, primitive operations of the interconnect network, and how it supports different communication schemes.

A. Requirements for Interconnect Network

The interconnect network should support unicast, multicast and broadcast as they are the most used communication schemes [13]. Unicast describes the communication between a single sender and a single receiver; multicast describes the communication between one (multiple) sender(s) and multiple receivers; broadcast describes the communication between one sender and all connected receivers. In this section, we show how these operations are realized in a memristor crossbar.

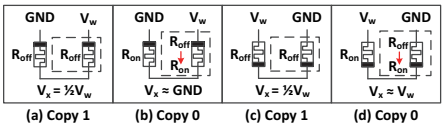


Fig. 2: Single-Fanout Copy Operation

B. Primitive Operations for Interconnect Network

The primitive operations are single- (one output memristor is involved) and multi- (multiple output memristors) fanout copy operations as depicted in Fig. 2 and 3, respectively. For each copy operation, we define the source memristor(s) as input memristor(s) and the target memristor(s) as the output memristor(s). In the figures, the output memristors are surrounded by a dash-lined square. All the voltages of the floating wires are denoted by V_x; V_x is used to explain the switching of the output memristor. Prior to a copy operation, each output memristor should be initialized to high resistance.

The single-fanout copy (SFC) was proposed by G. Snider in [7] and is shown in Fig. 2(a) and (b) for a copy 1 and 0, respectively. Fig. 2(c) and (d) show them for reversed memristor orientations, i.e., with reversed positive and negative terminals. In Fig. 2(a) and (b), *GND* is applied to the positive terminal of the input memristor, while V_w to the output memristor. In case the resistance of the input memristor equals R_{off} (see Fig. 2(a)), $V_x=1/2V_w$, and therefore, the voltage across the output memristor is $V_w-V_x=1/2V_w < V_{th}$. As a result, the output memristor stays at R_{off}. In case the resistance of the input memristor equals R_{on}, $V_x\approx$*GND*, and therefore, the voltage across the output memristor is $V_w-V_x\approx V_w > V_{th}$. As a result, it switches to R_{on}. Similar conclusions can be made for Fig. 2(c) and (d).

To support multicast and broadcast, we propose a novel multi-fanout copy (MFC) operation as shown in Fig. 3. It shows MFC for both memristor orientations (Fig. 3(a)-(c) and 3(d)-(f)). In Fig. 3(a)-(c), *GND* is applied to the positive terminal of the input memristor while V_w to the output memristor. In case, the resistance of the input memristor equals R_{off} (see Fig. 3(a)), $V_x=3/4V_w$. Therefore, the voltage across the output memristor is $V_w-V_x=1/4V_w<V_{th}$. As a result, the output memristor stays at R_{off}. In case the resistance of the input memristor equals R_{on} (see Fig. 3(b)), $V_x\approx$*GND*. Therefore, the voltage across each output memristor is $V_w-V_x\approx V_w>V_{th}$. As a result, the output memristors switch to R_{on}. However, MFC is a destructive operation. After the output memristors switch to R_{on}, the voltage across the input memristor changes to $V_x=3/4V_w>V_{th}$. As a result, the input memristor switches to R_{off} (see Fig. 3(c)). Similar conclusions can be made for the reversed memristor orientation in Fig. 3(d)-(f).

C. Support for Unicast, Multicast and Broadcast

Unicast communication schemes can be realized in memristor crossbars using SFC operations. Fig. 4 shows the unicast communication scheme for different cases. In the figure, memristors are represented by arrows with their positive terminal presented by the arrow head. The sender(s) and receiver(s) are

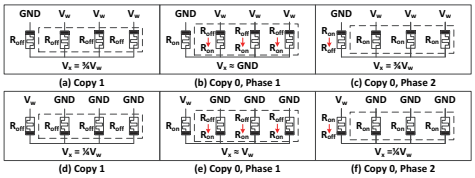


Fig. 3: Multi-Fanout Copy Operation

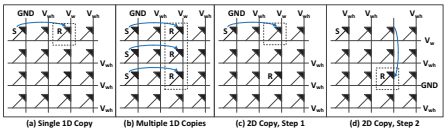


Fig. 4: Unicast Communication Schemes

marked by a letter S and R, respectively. Arch arrows present the direction of the copy operation. The output memristor of each copy operation is surrounded by a dash-lined frame work. Each memristor represents a single bit.

The unicast communication can be subdivided into one- and two-dimensional transmission. One-dimensional unicast transmission involves data transfer between a sender and receiver within the same row (horizontal) or column (vertical). Fig. 4(a) shows a horizontal unicast transmition as an example since the similar operation can be used for vertical unicast transmission. A one-dimensional unicast transmission requires an SFC operation in the same row (column). Multiple one-dimensional unicast transmissions can occur simultaneously as their floating nanowires do not interfere. Fig. 4(b) shows multiple horizontal unicast transmissions as an example. Two-dimensional unicast transmission involves data transfer between a sender and receiver that are not in the same row or column. A two-dimensional unicast transmission requires two sequential SFC operations. First the data is copied from the sender to an intermediate memristor in the same row (column) (see Fig. 4(c)), and subsequently from the intermediate memristor to the receiver in the same column (row) (see Fig. 4(d)). The transmission can be horizontal-first or vertical-first. The voltages *GND* and V_w are applied to the positive terminals of the input and output memristor, respectively when horizontal communication is used (see also Fig. 2 (a) and (b)). Oppositely, for vertical communication, V_w and *GND* are applied to the negative memristor terminals (see Fig. 2 (c) and (d)). All the other crossbar nanowires that are not involved in data transfer are set to V_{wh}. This prevents undesired switching of idle memristors and alleviates the impact of sneak path currents during the transmission [5].

Similar to unicast communication, multicast and broadcast communication schemes can be realized in the memristor crossbar by using MFC operations instead of SFC operations. As the working principle of unicast and multicast/broadcast implementation on memristor crossbar are similar, we do not describe multicast and broadcast operations in detail. Fig. 5 shows the multicast and broadcast communication schemes for different sender and receiver configurations.

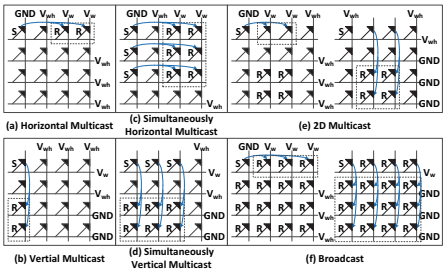


Fig. 5: Multicast and Broadcast Communication Schemes

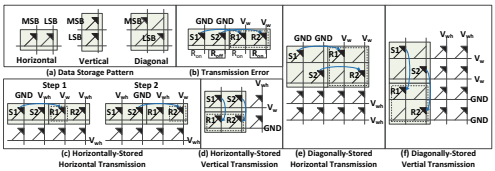


Fig. 6: Multi-Bit Word Storage Patterns and Communication

D. Impact of Multi-Bit Words

In digital circuits, each memristor stores one bit. To store multi-bit numbers, or words, three different patterns can be considered: horizontal, vertical and diagonal as shown in Fig. 6 (a) with green boxes (for a two-bit example). The horizontal and vertical patterns store a word in a row and column, respectively. The diagonal pattern stores a word diagonally. The remaining off-diagonal memristors in the green box are disabled [7] and have a permanent high resistance.

Based on the data storage pattern and communication direction, transmitting data in parallel might not always be feasible. For example, in Fig. 6 (b) the transmission of both bits simultaneously cannot work as both copy operations share the same floating nanowire. This causes an erroneous transmission on one of the output bits (Sender S2 has value R_{off}, while receiver R2 R_{on}). Therefore, the horizontal transmission of words must be performed sequentially bit-by-bit as depicted in Fig. 6 (c). In step 1, only the first input is transmitted, while in step 2 the second bit is transmitted. However, the vertical transmission of horizontally-stored words can be performed simultaneously as the copy operations do not share the same floating nanowire; this is depicted in Fig. 6 (d). Oppositely, vertically-stored words can transmit data simultaneously during horizontal transmission and require multiple steps for vertical transmission.

Fig. 6 (e) and (f) show horizontal and vertical transmission for diagonally-stored words, respectively. Both bit-copy operations can be performed simultaneously as they do not share the floating nanowires. Note that this storage requires disabled memristors on the off-diagonal positions which do not impact the transmission, but increases the area.

As the word size increases, the transmission time for horizontally- or vertically-stored data increases linearly (based

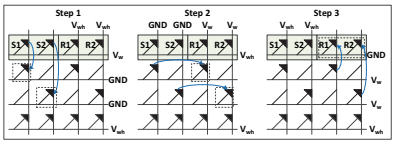


Fig. 7: Multi-Bit Communication with Mirrors

on the transmission direction) and may lead to high communication overhead. To solve this problem, we propose a communication scheme consisting of 3 steps (independent of the width of the word) using two mirrors. The *mirrors* are reserved storage units that use the diagonal data pattern. For horizontally-stored data (see Fig. 7), data is copied vertically to the first mirror in step 1; in step 2 this data is transferred horizontally to the second mirror; finally, in step 3 the data is transferred to the destination. Similarly, but with opposite directions, three steps are required for vertical transmission.

IV. CASE STUDY

This section describes three case studies: interconnect networks for (a) specific function within a tile (i.e., matrix transpose), (b) general logic functions within a tile and (c) communication between tiles.

A. Specific Intra-Tile Interconnect Network

The matrix transpose is a basic and one of the most used operations in matrix computation. Typically, this operation is very costly in traditional architectures [14]. We propose an intra-tile interconnect network to implement a matrix transpose operation. Fig. 8 (a) shows an example for a 2×3 matrix. Each word a_{ij} consists of 2 bits ($a_{ij}(1)$ presents MSB and $a_{ij}(0)$ presents LSB) stored in a diagonal pattern (see Fig. 8(b)). The words in the intermediate and final results are also stored in this pattern.

The proposed interconnect network consists of an input, intermediate and output blocks. The input and output blocks store the original and transposed matrix, respectively. The intermediate block stores one column of the input matrix at a time and subsequently, the transposed column is transferred to the output block. This is repeated for all the input columns. In this procedure, the intermediate block is referred to as a mirror.

The complete working procedure to transpose a matrix is captured by the state machine in Fig. 8(c); here en presents the enable signal for control logic, Row_cnt the number of rows which have been transposed, Row_num the total number of rows. State INI initializes the memristors of mirror and output block to R_{off} and write input matrix to input block. State IM2M (input matrix to mirror) copies a single column of the input matrix to the mirror based on the value of Row_cnt (see Fig. 8(d)); State M2OM (mirror to output matrix) copies the data from the mirror to the output matrix (see Fig. 8(e)); State INIM initializes the memristors of mirror to R_{off} for the next column. The control logic repeats state IM2M, M2OM and INIM until all the columns have been transposed.

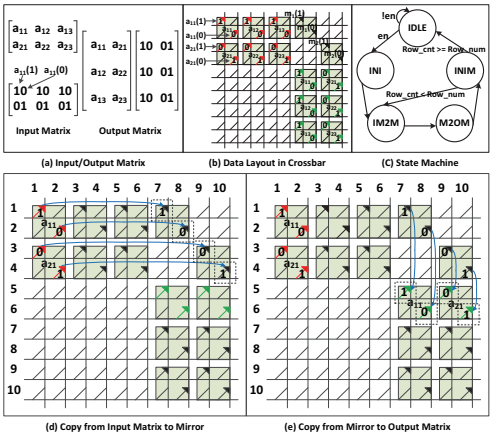


Fig. 8: Matrix Transpose on Memristor Crossbar

B. General Intra-Tile Interconnect Network

Based on single- and multi-fanout copy operations, an intra-tile interconnect network for general logic functions is proposed; a logic function including five logic blocks (i.e., A, B, C, D and E) is shown as an example in Fig. 9. The interconnect between different logic blocks of the block diagram depicted in Fig. 9(a) is implemented on the memristor crossbar using mirrors as depicted in Fig. 9(b). In the figure, the outputs of Block A and B (denoted by O_i) are transferred to the corresponding inputs of Block C, D and E. In Fig. 9 (b), each square with a diagonal line presents a mirror. The blue lines present the interconnects between the blocks similar as Fig. 9 (a). The red lines around each block present isolations between blocks [7] (e.g., breaking the horizontal and vertical nanowires). This guarantees that the blocks can work in parallel. As a memristor crossbar only contains horizontal and vertical nanowires, each interconnect between an input and output involves two mirrors where one is aligned with the input and one with the output. Both mirrors are placed on the same horizontal nanowires. When both the output and input of an interconnect reside in the same column, no mirrors are required and only a direct copy operation can transmit data. Otherwise, each transmission requires three sequential copy operations: 1) from the output to the first mirror, 2) from the first mirror to the second mirror, and 3) from the second mirror to the input. Outputs with multiple fanouts (e.g., O_5 in Fig. 9(b)) require a MFC operation.

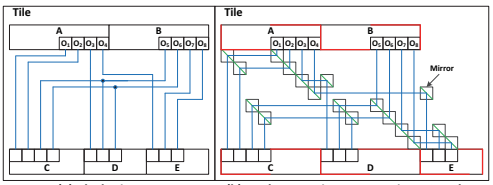


Fig. 9: General Intra-Tile Interconnect Network Using Mirrors

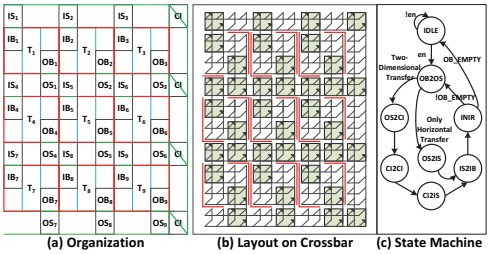


Fig. 10: 2D Bus General Inter-Tile Interconnect Network

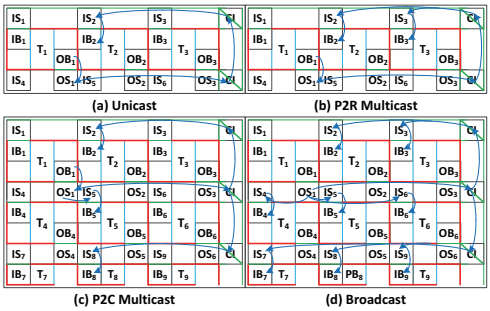


Fig. 11: 2D Bus Communication Schemes

C. General Inter-Tile Interconnect Network

We propose an interconnect network, 2D bus, between multiple processing blocks (PB) or tiles (T) as shown in Fig. 10. Each tile implements a complex logic function as proposed in [7]. Our 2D bus enables both horizontal and vertical transmission. It consists of input buffers (IB), output buffers (OB), input stops (IS), output stops (OS) and channel intersections (CI). The IB temporarily buffers the data received from 2D bus, while the OB temporarily buffers the data that should be transmitted via the 2D bus. These buffers allow the 2D bus and tiles to run without blocking each other. Each tile has its own dedicated an IB, OB, IS and OS. Horizontal transmission is handled by OS and IS and their task is to buffer and transmit data horizontally. The vertical transmission is handled by CI and its task is to switch the communication direction from horizontal to vertical and vice versa.

Fig. 10 (b) shows the crossbar implementation of a 2D bus. Note that details of tiles have been left out for convenience. All IBs, OBs, ISs, OSs and CIs employ mirrors to realize two-dimensional transmission within constant time (see Section III). Each tile is isolated from its neighbouring tiles that are in the same row and column (by breaking the horizontal and vertical nanowires shown by the red lines in Fig. 10 (a) and (b) [7]). Each tile communicates with its corresponding IS and OS via its IB and OB, respectively. Therefore, tiles do not influence each other.

Fig. 10 (c) shows the state machine. In the first state (OB2OS) the outputs of the sender tile are moved from OB to OS. In the second state, the data is moved from OB to IS if only horizontal transmission is involved (OS2IS). Otherwise, the data is moved to the CI that is on the same row of OS (OS2CI). The next state (CI2CI) moves the data vertically to the CIs which are on the same rows as the ISs of the receivers. Afterwards, data is moved horizontally from CI (CI2IS, if vertical transmission is involved) or OS (OS2IS, in case only horizontal transmission is involved) to the IS of the receiver. Next, data is moved from the IS to IB of the receiver (IS2IB). Finally, all memristors of IS, OS and CI are programmed to R_{off} in state INIR for the next transmission.

The 2D bus is able to support unicast, multicast, and broadcast communication as shown in Fig. 11. The figure shows each transfer step in an (blue) arch arrow. In all cases Tile_1 is assumed to be the sender. For the multicast operation, we consider two examples; they are point-to-row (P2R) and point-to-column (P2C) multicast (see Fig. 11 (b) and (c)). Most states of the state machine of Fig. 10 (c) execute the same operations for each communication scheme. The only states that differ CI2CI and CI2IS as they transfer to either multiple receivers on different rows or multiple receivers on the same row. The CI that is positioned in on the same row of the sender OS is referred to as the sender CI, while the CI on the same row of the receiver IS is referred to as the receiver CI.

The unicast (see Fig. 11(a)) involves SFC operations only. The P2R multicast (see Fig. 11(b)) is similar to the unicast except that the receiver CI uses a MFC operation to transmit data simultaneously to the receivers. The P2C multicast (see Fig. 11(c)) moves the data horizontally to the sender CI. Subsequently, it uses a MFC operation to send the data to the receiver CIs. Finally, the data is transmitted in parallel to all the receivers on the desired columns. Note that in multicast and broadcast the destructive MFC operation only affects the sender CI and has no impact on the correctness of the transmission. The broadcast (see Fig. 11(d)) is similar as P2C multicast, except that the last step consists of copying data from CI to all the receivers. However, there is major difference. As the sender CI may lose its data when its data is copied to other CIs. The P2C multicast (e.g.,OS_1 to IS_5 in Fig. 11(c)) and broadcast (OS_1 to IS_4, IS_5 and IS_6 in Fig. 11(d)) uses the sender OS to transmit data to its neighbours in the last step.

V. SIMULATION RESULTS

This section presents the simulation results of the interconnect network and the three communication schemes of Section IV (see Fig. 11); the delay and area are evaluated.

A. Simulation

The simulation framework consists of three parts; they are the memristor crossbar, the voltage drivers and their control logic [12]. The control logic determines the voltage that should be applied to the nanowires and the voltage drivers are used to drive them (either by applying a voltage or leaving them floating). The memristor crossbar is described by a SPICE netlist. The THAB model, control logic and voltage drivers are described in Verilog-A. Table I lists the simulation parameters. As we forcus on the functionality of the interconnect network,

TABLE I: Simulation Parameters

Memristor Model		Control Voltages	
Parameter	Value	Parameter	Value
R_{on}	100Ω	V_w	1.6V
R_{off}	200kΩ	V_{wh}	0.8V
V_{th}	1V		

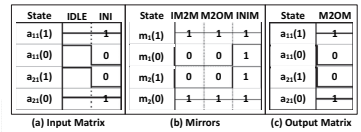


Fig. 12: Simulation Results of Matrix Transpose

the resistance and capacitance of nanowires are negelacted. Half-select voltage strategy is used to alleviate the impact of sneak path currents [5].

Fig. 12 shows the results of the simulated matrix transpose operation of Fig. 8. The input matrix is initialized in state INI. Mirrors receive and send a column of the input matrix in state IM2M and M2OM, respectively. Afterwards, mirrors are initialized to high resistance in state INIM. The output matrix receives the transposed column of the input matrix as a row.

The simulated communication schemes (i.e., unicast, P2R and P2C multicast, and broadcast) supported by a 2D bus are depicted in Fig. 11. Fig. 13 shows the SPICE simulation results. All the simulations use Tile_1 as the sender and send the same data 01. As the simulation intends to verify the communication only, no tiles are involved and data is directly written into the OB_1 of T_1 in state IDLE. The figure shows for the most important signals of all receivers (depending on communication schemes). All of them match with Section IV.

B. Evaluation

The proposed interconnect network for matrix transpose and 2D bus are evaluated using delay and area as metrics. Data for both applications is assumed to be stored in the diagonal pattern. Table II summarizes the results. For the matrix transpose we assume a single (grid size equals 1) matrix of size $m \times n$. Its delay is linear with the number of rows while the area equals. Each row requires 3 cycles to complete. Its area consist of three parts which are the input, intermediate and output blocks. The area grows linearly with the matrix size ($m \times n$) and quadratically with the word size; this is due to the diagonal storage pattern. A horizontal or vertical storage pattern reduces this quadratic to a linear dependency.

For the 2D bus, we assume a grid size of $A \times B$ (i.e., the system consists of $A \times B$ tiles whose size are $m \times n$). The delay of each operation is either 4 (only horizontal communication) or 6 (horizontal and vertical communication) cycles (see Fig. 10). The area of the bus is calculated by considering the whole area of the system minus the area of the $A \times B$ tiles.

In our simulations, the resitance and capacitance of nanowires are ignored. Therefore, the the number of communication steps might be too optimistic for crossbars arrays with long nano-wires, especially for the horizontal crossbar lines as they

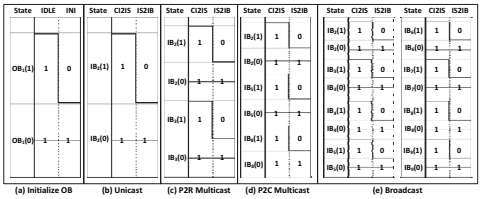


Fig. 13: Simulation Results of 2D Bus

TABLE II: Evaluation

	Matrix Transpose	2D Bus
Conditions		
Word Size	b	
Grid Size	1	$A \times B$
PB Size	$m \times n$	$m \times n$
Metrics		
Delay	$3m$	4 or 6
Area	$2mnb^2 + m^2b^2$	$\lceil B(n+2b)+b \rceil \lceil A(m+b)+b \rceil - ABmn$

contain only a single CI. In practice, additional repeaters might be required for proper operation. They can be realized straightforwardly by inserting additional CI channels (on the horizontal crossbar lines) or intermediate storages. Nevertheless, these repeaters do not impact the principle of operation.

VI. CONCLUSION

Memristor circuits are a promising candidate for the next-generation VLSI circuits. In these VLSI circuits, the interconnection plays a vital role. To our knowledge, this paper is the first to address scalable interconnect networks for memristor crossbar arrays. It realizes this by using reserved memristors and supports both intra- and inter-tile communication. This approach is verified using SPICE simulations.

REFERENCES

[1] B. Hoefflinger, "The energy crisis," in *Chips 2020*, ser. The Frontiers Collection, B. Hoefflinger, Ed., Jan. 2012, pp. 421–427.
[2] W. Zhao *et al.*, "Nanodevice-based novel computing paradigms and the neuromorphic approach," in *ISCAS*, May 2012, pp. 2509–2512.
[3] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.
[4] P. Mazumder *et al.*, "Memristors: Devices, models, and applications," *Proceedings of the IEEE*, vol. 100, pp. 1911–1919, 2012.
[5] K.-H. Kim *et al.*, "A functional hybrid memristor crossbar-Array/CMOS system for data storage and neuromorphic applications," *Nano Letters*, vol. 12, pp. 389–395, Jan. 2012.
[6] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015, p. 199.
[7] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.
[8] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.
[9] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, pp. 2632–2663, 2009.
[10] W. Lu *et al.*, "Two-terminal resistive switches (memristors) for memory and logic applications," in *ASP-DAC*, Jan. 2011, pp. 217–223.
[11] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, pp. 211–221, 2013.
[12] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *VLSI, IEEE Transactions on*, vol. 22, pp. 2054–2066, 2014.
[13] A. S. Tanenbaum, "Computer networks 4th edition," *ed: Prentice Hall*, 2003.
[14] G. H. Golub *et al.*, *Matrix computations*. JHU Press, 2012, vol. 3.

5

CIRCUIT DESIGN AND SYNTHESIS FLOW

This chapter presents design methodologies for both ASIC and FPGA using memristor logic design styles. First, it explores place-and-route algorithms for large scale memristor crossbar ASIC design; a synthesis flow and evaluation model are proposed as well. Subsequently, it explores different FPGA implementations using two different logic design styles and proposes a synthesis flow and evaluation model.

The content of this chapter consists of the following research articles:

1. ***L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Volume 37, Issue 2, Feb 2018.*

2. ***L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, M Alfailakawi, Non-Volatile Look-up Table Based FPGA Implementations, IEEE International Symposium on Design and Test (IDT), Hammamet, Tunisia, December, 2016, pp. 165-170*

5.1. INTRODUCTION

To implement resistive computing architectures, the large scale logic circuits are required. Logic circuits normally can be divided into application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) depending on the flexibility and reconfigurability of the circuits. Using the primitive gates and interconnect networks presented in Chapter 3 and 4, this chapter explores circuit design methodologies for both ASICs and FPGAs.

ASIC Design Methodology: Boolean logic seems to be the enabler for the resistive computer architectures as they use resistance to represent data, and can be easily integrated with high density memories [59,60]. In [22,25], the authors proposed *simple* and *small* Boolean logic designs for memristor crossbar, which partially address the shortcomings of implication logic. Therefore, exploring memristor crossbar logic design for the larger circuits is required in order to appropriately assess the potential of memristor technology.

FPGA Design Methodology: Many novel memristor-based FPGAs [46,61,62], called MemFPGA for short, have been reported recently. MemFPGAs typically employ the classical island-style architecture [63] which consists of configurable logic blocks (CLBs), programmable interconnect, and block RAM (BRAM). Each CLB consists of look-up tables (LUTs) and a D flip-flop (DFF). Both CLBs and the programmable interconnect use memories to store configuration information. In MemFPGAs, memristors are utilized in the following fashions: 1) configuration memories for CLBs and programmable interconnects [61], 2) implementation of programmable interconnect [46], and 3) implementation of BRAMs and DFFs [62]. However, no LUTs have been implemented by memristor logic circuits.

5.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

1. *ASIC Design Methodology* [25,56]: In order to implement large scale ASICs, the place-and-route approaches are explored first. As the memristive devices are passive, a CMOS-based peripheral circuit design is proposed to control them. Next, we propose several optimization techniques to improve the performance of Snider logic in terms of area, delay and energy. In order to estimate the performance of logic circuits, an evaluation model is proposed while considering both the memristor crossbar and CMOS controller. Finally, we present a synthesis flow by adapting conventional CMOS synthesis flows. To illustrate the potential of the proposed design methodology, a multi-bit adder and other nine more complex benchmarks are studied; the delay, area and power consumption induced by both the memristor and its CMOS control part are evaluated. The results show that optimization techniques can improve the area and delay in the factor of 7.8 and 2.2 at least. In addition, the circuit scalability and the impact of technology parameters on delay and power consumption are also discussed. It is observed that the area of the whole circuit can be dominated by either crossbar array or CMOS controller de-

pending on the scale of circuits; the faster resistive switching and higher on-state resistance value of memristor device can significantly reduce the delay and power consumption of the circuits.

2. *FPGA Design Methodology* [47]: Based on the classical island-style FPGA architecture, we use two different logic design styles to implement the look-up tables. In order to appropriately estimate the performance of the proposed FPGA implementations, we propose an evaluation model which considers both the memristor and CMOS parts. Finally, a synthesis flow is developed by adapting CMOS based FPGA synthesis flows. To illustrate the potential of our proposals, they are benchmarked using Toronto 20 benchmark suite, and compared with the state-of-the-art in terms of area and delay. The experimental results show that both the area and delay of the novel FPGAs are improved up to 4.24x and 1.46x respectively.

A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering,
Delft University of Technology, Delft, the Netherlands
Email: {L.Xie,H.A.DuNguyen,M.Taouil,S.Hamdioui,K.L.M.Bertels}@tudelft.nl

***Abstract*—Alternatives to CMOS logic circuit implementations are under research for future scaled electronics. Memristor crossbar based logic circuit is one of the promising candidates to at least partially replace CMOS technology, which is facing many challenges such as reduced scalability, reliability, and performance gain. Memristor crossbar offers many advantages including scalability, high integration density, non-volatility, etc. The state-of-the-art for memristor crossbar logic circuit design can only implement simple and small circuits. This paper proposes a mapping methodology of large Boolean logic circuits on memristor crossbar. Appropriate place-and-route schemes, to efficiently map the circuits on the crossbar, as well as several optimization schemes are also proposed. To illustrate the potential of the methodology, a multi-bit adder and other nine more complex benchmarks are studied; the delay, area and power consumption induced by both the crossbar and its CMOS control part are evaluated.**

***Index Terms*—Memristor Crossbar, Logic Design, Mapping, Evaluation.**

I. INTRODUCTION

As CMOS transistors gradually scale down to the intrinsically physical device limits, CMOS VLSI circuits are facing major challenges such as saturated performance gain, increased leakage power consumption, reduced reliability, and a more complex fabrication process [1–3]. In addition, CMOS-based computers are suffering from memory bottleneck [4], power wall [5], etc. To address these challenges, alternative technologies [6] are under investigation; examples are nanotube [7,8], silicon nanowire FET [9], magnetic/spintronic [10–12], and memristors [13,14]. Among these proposals, memristor crossbar based logic circuit is a promising candidate due to its attractive characteristics in terms of scalability, high integration density, and non-volatility, etc [15,16]. Moreover, based on memristor technology, novel computer architectures for data-intensive applications have been proposed, such as computation-in-memory [17–21], resistive associate processor [22] and Pinatubo [23]; they show a potential of order of magnitude performance improvement as compared to todays' architectures.

To implement such novel computer architectures, logic circuits based on resistive devices, such as memristors, are required; research on this topic is still in infancy stage. As of today, four types of memristor-based logic circuits have been proposed: threshold [24,25], majority [25], material implication [26,27], and Boolean [28,29] logic. Threshold and majority logic circuits use memristor as the weight of inputs and use CMOS current mirror or inverter as the threshold function. Both of them are more suitable for traditional computer architecture as they represent data by using voltage. In contrast, both material implication and Boolean logic seem to be the enabler for the novel computer architectures as they use resistance to represent data, and can be easily integrated with high density memories [30,31]. In [27,32,33], the authors proposed methodologies to implement logic functions using a sequence of material implication operations. However, these methodologies suffer from low speed and require new algorithms to implement arithmetic operations such as addition [27,34,35]. In [28,29], the authors proposed *simple* and *small* Boolean logic designs for memristor crossbar, which partially address the shortcomings of implication logic. Therefore, exploring memristor crossbar logic design for larger circuits is required in order to appropriately assess the potential of such technology.

This paper proposes a mapping methodology of Boolean logic circuits on memristor crossbar to enable the implementation of large logic circuits, and illustrates the methodology for a multi-bit adder. Thereafter, the methodology is applied to nine more complex benchmarks to show its generality. This work is built on our preliminary work published in [29], where the focus was mainly on the implementation of simple Boolean functions. Compared to the preliminary work, the new contributions of this paper are:

- A mapping flow for memristor crossbar enabling large-scale logic circuits.
- Two place-and-route schemes to map large-scale logic circuits on crossbar.
- Design of CMOS peripheral circuits, which act as the control engine of the memristor crossbar.
- Several schemes to optimize the area, delay and power consumption.
- A model to evaluate the performance of the design in terms of area, delay and power consumption, which considers both the crossbar and CMOS parts.

The remainder of this paper is organized as follows. Section II briefly describes the design of resistive Boolean logic. Section III presents the proposed mapping flow, two place-and-route schemes and CMOS circuits to control the crossbar. Section IV discusses several optimization schemes. Section V verifies the methodology using multi-bit adders as a case study, and

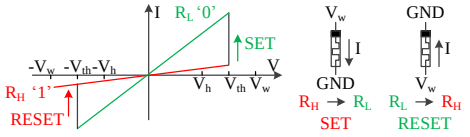


Fig. 1: Ideal Memristor Model.

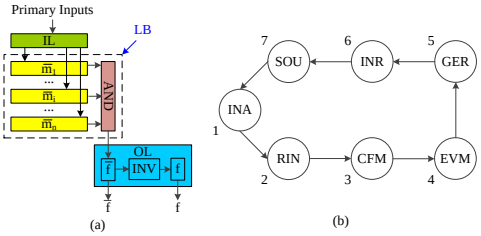


Fig. 2: Working Principle of Boolean Logic: (a) Computing Element, (b) State Machine.

applies the method to nine more complex benchmarks. Section VI concludes the paper with advantages of the proposal and challenges in the future work.

II. Boolean Logic Design

This section starts first by briefly presenting the memristor model used in this work. Thereafter, it presents the working principle of the resistive Boolean logic we proposed in [29]. Then, the implementation of the primitive (logic) operations (e.g., AND) is given; these are used to build one-bit full adder, which is used in this work later.

A. Memristor Model

The left part of Fig. 1 shows the current-voltage relation of the ideal memristor model used in this work; it has a high (R_H) and low (R_L) resistive states. The memristor switches from one resistive state to another when the absolute value of the voltage across the device is greater than its threshold voltage V_{th}. Otherwise, it stays in its current resistive state. Typically, a memristor requires two different switching threshold voltages to switch from low to high resistance (RESET) and from high to low resistance (SET) [36,37] (see the right part of Fig. 1). For simplicity, we assume that the threshold voltage V_{th} (in absolute value) for both switchings are the same. Here, we use the ideal model as this paper focuses on mapping methodology. Nevertheless, any model can be used such as those in [36,37].

B. Working Principle of Boolean Logic

Our Boolean logic design approach [29] is able to implement any logic function f expressed in the format of sum-of-product (i.e., $f = m_1 + \cdots + m_i + \cdots + m_n = \overline{m_1} \cdot \cdots \cdot \overline{m_i} \cdot \cdots \cdot \overline{m_n}$ where m_i is a minterm of inputs, n the number of minterms); its implementation is referred to as a computing element (CE) as shown in Fig. 2(a). A CE consists of an input latch (IL), an output latch (OL), and a logic block (LB). The LB consists

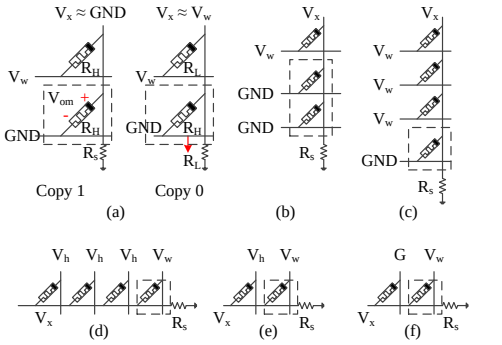


Fig. 3: Implementation of Primitive Operations: (a) Single-Fanout Copy, (b) Multi-Fanout Copy, (c) 3AND1, (d) 3NAND1, (e) INV1, (f) Horizontally Copy.

of all the minterms of the Boolean function; each $\overline{m_i}$ is realized using a NAND gate consisting of several memristors depending on the number of its inputs. By ANDing all the $\overline{m_i}$, the $\overline{f}$ can be generated. Finally, $\overline{f}$ is inverted to obtain f. The input and output latches are composed of several memristors depending on the number of inputs and outputs of the Boolean function, respectively.

Memristor-based logic design described above requires a CMOS circuit to control the crossbar part; its behaviour is captured by a state machine as shown in Fig. 2(b). The state machine requires 7 states:

- INA: INitialize All the memristors of a CE to R_H. This state requires RESET operations.
- RIN: Receive INputs. The IL of the CE receives the inputs from primary inputs using CMOS controller to program the resistance of memristors, or from the OL of the previous CE using copy operations. Therefore, this state requires SET, RESET, or copy operations.
- CFM: ConFigure all Minterms. All the minterms are configured simultaneously through copying inputs stored in IL to each minterm in parallel. Hence, this state requires copy operations.
- EVM: EValuate all Minterms. All the $\overline{m_i}$ are evaluated at the same time; each $\overline{m_i}$ is implemented by an NAND operation.
- GER: GEnerate Result. The results of EVM are used as inputs of an AND gate to generate $\overline{f}$, which is the negation of the Boolean function. Therefore, this state needs an AND operation.
- INR: INvert Result. The result of GER is inverted to produce the final result f. Hence, an inversion (INV) is required.
- SOU: Send OUtputs. Finally the result stored in OL is sent to IL of the next CE. Hence, copy operations are employed.

The above shows that in order to implement Boolean logic using the described approach, at least five primitive operations are needed: RESET, copy, NAND, AND, and INV; RESET

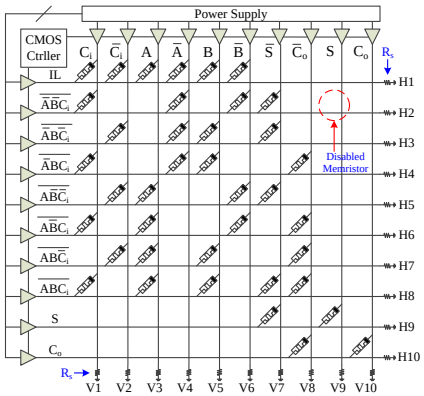


Fig. 4: Implementation of One-Bit Full Adder.

5

is already described and is shown in Fig. 1; the remaining operations are discussed next.

C. Implementations of Primitive Operations

All primitive operations use R_H and R_L to represent logic 1 and 0, respectively. Fig. 3 shows the implementations of all primitive operations; each implementation consists of one or multiple input and an output memristors. The output memristors are all initialized to R_H prior to any operation (i.e., RESET operation of state INA), and are surrounded by a dashed-line box in the figure. The voltage across the output memristor(s) is denoted by V_{om}, while the voltage of the floating nanowire is denoted by V_x; both of them are used to explain the working principle of primitive operations. In addition, all primitive operations consist of a resistor R_s (see Fig. 3), which satisfies the condition $R_L \ll R_s \ll R_H$; this is required to guarantee that the voltage across the output memristor is close to the desired voltage for proper operations [27]. Fig. 3(a), (c), (d) and (e) show primitive operations with one output memristor or single fanout. Multi fanout operations can be realized by employing multiple output memristors; Fig. 3(b) shows a two-fanout copy operation by employing two output memristors. Note that the positive terminal of each memristor is connected to the vertical nanowire. Fig. 3(f) shows the horizontal copy; it will be used later.

To control primitive operations, three different voltages are required: V_w, V_h, and *GND*; see Fig. 1. V_w is used to program the memristor; V_h is used to minimize the impact of sneak path currents by half-select voltage strategy [38]; V_h is then applied to memristors which are not involved in particular operations within a crossbar. V_h is also used to control NAND and INV as shown in Fig. 3(d) and Fig. 3(e). The relationship between V_w, V_h, *GND* and V_{th} is $0<V_h=\frac{V_w}{2}<V_{th}<V_w$. This relationship guarantees $V_w-V_h=2V_h-V_h=V_h<V_{th}$ which prevents undesired switching of non-accessed memristors [34,39].

The copy operation will be used as an example to explain its working principle; the other operations can be understood in a similar way, and more details can be found in [29]. Before performing any operation, the data should be stored in the right locations. Then, for a copy operation, a voltage $V_w>V_{th}$ and *GND* are applied to the input and output memristors, respectively, as shown in Fig. 3(a). In case of copy 1 (R_H), V_x is around 0 as $R_s \ll R_H$. Therefore, $V_{om}=V_x-0\approx 0<V_{th}$. As a result, the output memristor stays at R_H. In case of copy 0 (R_L), $V_x \approx V_w$ as $R_L \ll R_s$. Therefore, $V_{om}=V_x-0\approx V_w>V_{th}$. As a result, the output memristor switches to R_L.

TABLE I: Control Voltages for One-Bit Full Adder

States	Control Voltages					
	Row			Column		
	IL	LB	OL	IN	Output	
					OUT	OUTN
	H1	H2-8	H9-10	V1-6	V8,10	V7,9
INA	V_w	V_w	V_w	G	G	G
RIN	G	V_h	V_h	F	V_h	V_h
CFM	V_w	G	V_h	F	V_h	V_h
EVM	V_h	F	V_h	V_h	V_h	V_w
GER	V_h	V_w	G	V_h	V_h	F
INR	V_h	V_h	F	V_h	V_w	V_h
SOU	V_h	V_h	V_w	V_h	V_h	V_h

D. One-Bit Full Adder

The sum (S) and carry (C_o) of a one-bit full adder (FA) are expressed by Eq.1 [40]. Each equation consists of four minterms.

$$\begin{aligned} S &= \overline{\bar{A}\bar{B}C_i} \cdot \overline{\bar{A}B\bar{C}_i} \cdot \overline{A\bar{B}\bar{C}_i} \cdot \overline{ABC_i} \\ C_o &= \overline{\bar{A}BC_i} \cdot \overline{A\bar{B}C_i} \cdot \overline{AB\bar{C}_i} \cdot \overline{ABC_i} \end{aligned} \quad (1)$$

Fig. 4 shows the crossbar implementation of this FA using the principle of Fig. 2 and Fig. 3. For convenience, H# and V# are used to denote a horizontal and vertical nanowire, respectively; a memristor in the crossbar is denoted by M(H#,V#). To implement the FA, two types of memristors are used: active (which can switch between two resistive states) and disabled (which is permanently high resistance) memristor. In the figure, the junctions where disabled memristors are located have no devices as shown in Fig. 4.

The FA is implemented using a CE consisting of an IL, LB and OL. The IL is mapped on the memristors M(H1,V1-V6), since IL consists of primary inputs and their complements. The remaining memristors on H1 are disabled. The LB consisting of seven minterms is mapped on H2-H8, where the minterm $\overline{ABC_i}$ is shared by sum and carry. Each minterm is implemented by placing active memritors at junctions formed by the horizontal nanowire (representing the minterm) and (a) the vertical nanowires associated with the minterm's inputs, or (b) an output for which the minterm is part of. For example, $\overline{\bar{A}\bar{B}C_i}$ on H2 is a minterm of sum. Therefore, memristors M(H2,V1=C_i), M(H2,V4=$\bar{A}$), M(H2,V6=$\bar{B}$), and M(H2,V7=$\overline{S}$) are active; while the remaining memristors on H2 are disabled. The four minterms of S and those of C_o (see Eq. 1) are then ANDed in parallel by column V7=$\overline{S}$ and V8=$\overline{C_o}$, respectively. The OL is realized by H9 and H10. The

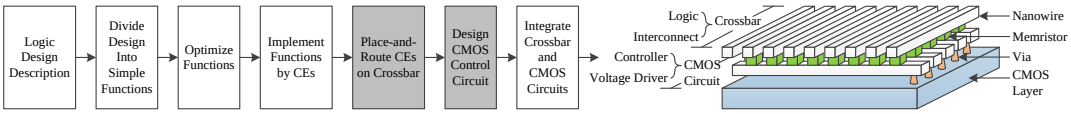


Fig. 5: Mapping Flow and Implementation.

results provided by the two ANDs are then stored at M(H9, V7) and M(H10, V8), which are thereafter inverted and stored at M(H9, V9=S) and M(H10, V10=C_o), respectively. Note the FA implementation requires 10 rows and 10 columns.

To perform the desired primitive operations during each state, appropriate voltages are applied to each horizontal and vertical nanowire of the CE. Table I summarizes the required voltages for the FA; they are straightforwardly derived from the implementations of the primitive operations as shown in Fig. 3. Each row (horizontal nanowire) is associated with the implementation of IL, LB or OL; while the columns are associated with the primary inputs (IN) or outputs (OUT), or their complements (OUTN). For instance, to perform copy operations required by CFM state to configure all minterms in parallel, V_w is applied to row H1, *GND* (G) is applied to rows H2-H8, while columns V1-V6 are left floating (F) (see the row CFM of Table I). It is worth noting that the remaining rows (H9-H10) and columns (V7-V10) are set to V_h in order to minimize the impact of sneak path currents [34,39]. All the circuits mentioned in this paper use this methodology to solve the sneak path problems.

III. Mapping Methodology and Implementation

This section first presents the mapping flow for Boolean logic based on memristor crossbar. Subsequently, it highlights the challenges of place-and-route within crossbar and potential solutions. Finally, it presents the CMOS circuit used to control the memristor crossbar.

A. Mapping Flow

Fig. 5 shows the flow of the mapping methodology. The entire design is first divided into multiple simple Boolean functions (e.g., look-up tables), which can be further optimized by EDA tools such as ESPRESSO [41]. Next, the optimized Boolean functions are implemented using CEs, as presented in Section II. Thereafter, all CEs are placed and routed within the crossbar, and the CMOS circuit (used to control the crossbar) is designed. Finally, the memristor crossbar and CMOS control circuits are integrated together by stacking the crossbar on the CMOS part as shown in Fig. 5. The first three steps are described in our previous work [29]; this section will focus on the place-and-route and CMOS circuit design.

B. Potential Solutions to Place-and-Route

To highlight the challenge of place-and-route, a 4-bit ripple carry adder is used as an example. Fig. 6(a) shows this 4-bit adder which uses four FAs of Fig. 4 as building blocks. A naive solution to place and route these FA blocks is to arrange them adjacently to each other, as the two options

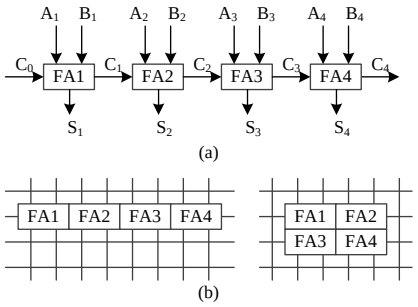


Fig. 6: Place-and-Route Challenges: (a) Block Diagram, (b) Layouts Sharing Nanowires.

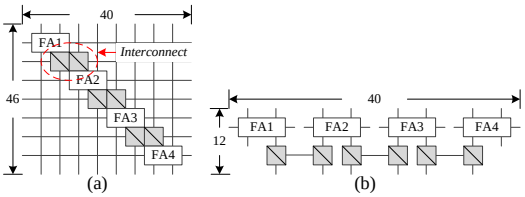


Fig. 7: Place-and-Route Schemes: (a) Diagonal Scheme, (b) Isolated Scheme.

shown in Fig. 6(b). However, typically this cannot be done as the operations of each FA require specific control voltages applied to the horizontal and vertical nanowires (see Table I); sharing these nanowires between different FAs will lead to a conflict of control voltages; and hence impacting each other's operations. Therefore, special attention should be given to place-and-route.

Potential solutions to address the above challenge are:

- Preventing each pair of FAs from sharing the same horizontal or vertical nanowires;
- Breaking the nanowires within the crossbar in order to isolate each FA;
- Stacking FAs on each other rather than having them within the same crossbar layer.

In the rest of this subsection, we will discuss the first two potential solutions in more details. Actually the third potential solution is similar to the second one except that the FAs are stacked.

To prevent each pair of FAs from sharing the same horizontal and vertical nanowires, diagonal place-and-route scheme is proposed. Fig. 7(a) shows the 4-bit adder which is placed and routed using the diagonal scheme. All the FAs are placed

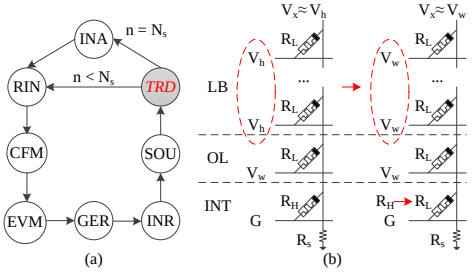


Fig. 8: CMOS Controller: (a) Modified State Machine, (b) State SOU.

TABLE II: Control Voltages for Multi-CE Design

State	Control Voltage						
	Row				Column		
	IL	LB	OL	INT	IN	OUT	OUTN
INA	V_w	V_w	V_w	V_w	G	G	G
RIN	G	V_h	V_h	V_h	F	V_h	V_h
CFM	V_w	G	V_h	V_w	F	V_h	V_h
EVM	V_h	F	V_h	V_h	V_h	V_h	V_w
GER	V_h	V_w	G	V_h	V_h	V_h	F
INR	V_h	V_h	F	V_h	V_h	V_w	V_h
SOU	V_h	V_w	V_w	G	V_h	V_h	V_h
TRD	V_h	V_h	V_h	F	V_w	G	G

in a diagonal pattern, and therefore, no FAs share the same horizontal and vertical nanowires. To route the carry and its negation between FAs (e.g., carry C_1 and it negation $\overline{C_1}$ between FA1 and FA2), two extra rows are reserved for interconnect [42]. As a result, the implementation of the four-bit adder of Fig. 6(a) is mapped on a 46×40 crossbar using the diagonal scheme; see Fig. 7(a).

Fig. 7(b) shows the 4-bit adder which is placed and routed using isolated scheme. By breaking the nanowires within the crossbar, each pair of FAs is isolated. Therefore, all the FA units can be placed adjacently, and the adder requires less crossbar area. Similar to the diagonal scheme, two extra rows are reserved for interconnect to route carry between FAs. The neighbour interconnect segments are isolated with each other, and each FA only connects to the interconnect segment connecting its up- and down-stream FAs. As a result, the implementation of the four-bit adder of Fig. 6(a) is mapped on a 12×40 crossbar using the isolated scheme; see Fig. 7(b). Note that the isolated scheme consumes less crossbar area than the diagonal scheme to place and route the same design; e.g., the crossbar area is reduced from 46×40 to 12×40; a reduction of 74%.

C. CMOS Control Circuits

To control the memristor crossbar, a CMOS circuit is employed; it consists of a controller and voltage drivers. The behaviour of the controller is captured by the state machine in Fig. 8(a); it is generated based on the state machine of Fig. 2(b). As the crossbar consists of multiple CEs, the state machine of Fig. 2(b) is extended with an extra state TRD (transfer data), which is needed to horizontally transfer

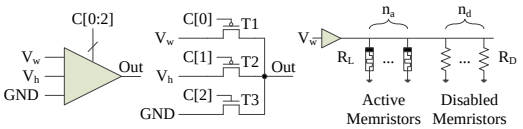


Fig. 9: CMOS Voltage Drivers.

the carry (using the interconnect) between FAs. Note that the execution of the state machine of Fig. 8(a) needs to be repeated N_s times after the initialization, where N_s is the number of stages (i.e., FAs) that the crossbar consists of. For example, N_s for the design of Fig. 6(a) is 4. Consequently, a design with N_s stages requires $7N_s$ + 1 execution steps, where 7 is the number of states (initialization excluded).

Table II summarizes the control voltages required for each state of Fig. 8; it is derived from Table I. As already mentioned, additional rows are needed for interconnects; they are included in the table and denoted as 'INT'. Note that the states in the table are extended with state TRD.

In addition, the control voltages of state CFM and SOU in Table II are different from Table I, due to the impact of sneak path currents [43]. Fig. 8(b) shows state SOU as an example. In state SOU, the data in OL is copied to INT; therefore, voltage V_w and *GND* are applied to row OL and INT, respectively. To reduce the impact of sneak path currents, V_h are typically applied to rows LB as half-select voltage [44]. Let assume that OL stores '0' (R_L). As OL stores the results of AND operations of state GER, at least one of the memristors in rows LB is R_L as shown in the left part of Fig. 8(b). After applying control voltages, the voltage of the floating nanowire V_x is around V_h, and the voltage across the output memristor in INT is $V_x - 0 \approx V_h < V_{th}$. As a result, the output memristor stays at R_H, and cannot copy '0'. To solve this issue, V_w is applied to row LB, and hence $V_x \approx V_w > V_{th}$; see the right part of Fig. 8(b). Consequently, the output memristor in INT switches to R_L, and copies 0 from OL.

Next, we will illustrate how to design a voltage driver while taking the restrictions of the crossbar design into account. Fig. 9 shows a possible implementation of voltage drivers, which are parts of the control circuit. A voltage driver is composed of one NMOS and two PMOS pass transistors; the state (i.e. closing or opening) of these transistors are controlled by three-bit signals C[0:2], which are provided by the CMOS controller. To drive a nanowire connecting active memristors as shown in the right part of Fig. 9, the transistors should provide enough current. Therefore, their width-to-length ratio $\frac{W}{L}$ should be carefully determined. Let assume that we have n_a active and n_d disabled memristors. To program a single active memristor, the current supplied by a NMOS transistor should be greater than $I_w = \frac{V_w}{R_L}$ [45]. Therefore, the NMOS transistor typically needs $(\frac{W}{L})_n = 2$ and its area has to be $A_n = 6F^2$ [46,47], where F is the feature size of CMOS technology;

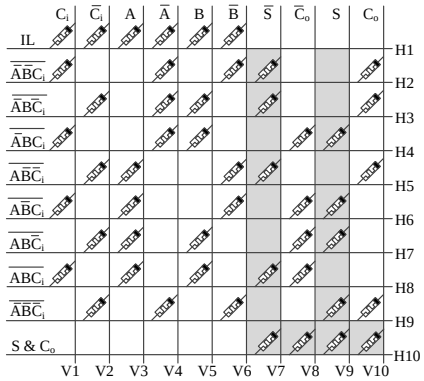


Fig. 10: The FA Calculating All Outputs Simultaneously.

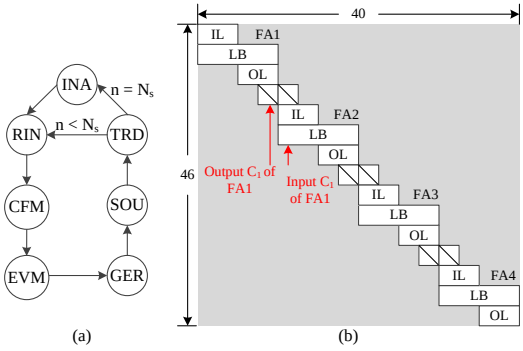


Fig. 11: Diagonally-Mapped Adder: (a) State Machine, (b) Layout.

see Eq. 2.

$$\begin{cases} I_{\rm w} = \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2, \\ A_{\rm n} = 6F^2 \end{cases} \tag{2}$$

To drive n_a active memristors in parallel, $(\frac{W}{L})_{\rm n}$ and $A_{\rm n}$ of the NMOS should be increased n_a times in order to provide the required current $I_{\rm w}$, as shown in Eq. 3.

$$\begin{cases} I_{\rm w} = n_a \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2n_a, \\ A_{\rm n} = 6n_a F^2 \end{cases} \tag{3}$$

In addition, assume that we have n_d disabled memristors; each of them consumes the current $I_{\rm D}$=$\frac{V_w}{R_D}$, and hence the NMOS transistor should be adjusted as to compensate for the current through n_d disabled memristors, as shown in Eq.4.

$$\begin{cases} I_{\rm w} = n_a \frac{V_w}{R_L} + n_d \frac{V_w}{R_D} = (n_a + \frac{R_L}{R_D} n_d) \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2(n_a + \frac{R_L}{R_D} n_d), \\ A_{\rm n} = 6(n_a + \frac{R_L}{R_D} n_d) F^2 \end{cases} \tag{4}$$

As the mobility of PMOS transistor is typically twice lower than that of NMOS [48], the $\frac{W}{L}$ of the PMOS has to be twice larger than that of NMOS. Therefore, the $(\frac{W}{L})_{\rm p}$ and area $A_{\rm p}$ of PMOS are obtained as expressed in Eq.5.

$$\begin{cases} (\frac{W}{L})_{\rm p} = 2(\frac{W}{L})_{\rm n} = 4(n_a + \frac{R_L}{R_D} n_d), \\ A_{\rm p} = 2A_{\rm n} = 12(n_a + \frac{R_L}{R_D} n_d) F^2 \end{cases} \tag{5}$$

Finally, the total area $A_{\rm vd}$ of a single voltage driver which consists of one NMOS and two PMOS pass transistors is given in Eq.6. Typically, $\frac{R_D}{R_L} > 5 \times 10^4$ [49]; hence, the number of active memristors n_a dominates the area of the voltage driver.

$$A_{\rm vd} = A_{\rm n} + 2A_{\rm p} = 30(n_a + \frac{R_L}{R_D} n_d) F^2 \approx 30 n_a F^2 \tag{6}$$

IV. MAPPING OPTIMIZATION SCHEMES

This section presents three mapping schemes to optimize the delay and/or area of a memristor crossbar logic design. These schemes can be used separately or in a combination.

A. Scheme 1: Calculate All Outputs Simultaneously

The first optimization scheme is calculating both the primary and complementary outputs at GER state. For example, the FA of Fig. 4 is optimized as shown in Fig. 10; its outputs S and $\overline{S}$ are expressed by Eq.7.

$$\begin{aligned} S &= \overline{\bar{A}B\bar{C}_i} \cdot \overline{\bar{A}\bar{B}C_i} \cdot \overline{AB\bar{C}_i} \cdot \overline{A\bar{B}\bar{C}_i} \\ \overline{S} &= \overline{\bar{A}B\bar{C}_i} \cdot \overline{\bar{A}\bar{B}C_i} \cdot \overline{A\bar{B}\bar{C}_i} \cdot \overline{ABC_i} \end{aligned} \tag{7}$$

All the eight required minterms of output S and $\overline{S}$ are implemented by placing active memristors on related junctions in a similar manner as Fig. 4. Therefore, all the minterms of both S and $\overline{S}$ are calculated at state EVM, and then ANDed to obtain the output S and $\overline{S}$ at state GER. The similar approach can be applied to the output C_o and $\overline{C_o}$; see Fig. 10. In addition, the OL can be implemented by only a single row, as $\overline{S}$ and $\overline{C_o}$ are calculated at state GER, rather than by inverting S and C_o as in Fig. 4(a). As both the primary and complementary outputs are obtained at state GER, state INR of Fig. 8(a) can be removed as shown in Fig. 11(a); hence, the four-bit adder of Fig. 7 (a) reduces the number of execution steps from $7N_s$+1=29 to $6N_s$+1=25. Fig. 11(b) shows the layout of the new four-bit adder implementation, where the FAs of Fig. 4 have been replaced with the FA of Fig. 10. Note that this new implementation requires the same area (i.e., 46×40) as the initial design of Fig. 7(a).

However, this scheme typically requires more area as all the $2^{\rm n}$ minterms of an n-input Boolean function must be mapped on the crossbar, instead of the required minterms only. For instance, the output S only needs four minterms as shown in Eq.7, and they can be implemented by only four rows in the crossbar. As both primary and complementary outputs are required as the inputs of the next FA stage, the output $\overline{S}$ should also be calculated, and hence, all eight minterms should be implemented by eight rows of the crossbar. To alleviate the incurred area overhead, several Boolean functions that have the same minterms can be implemented using the same share of hardware. For instance, the sum and carry of a FA are implemented together as shown in Fig. 10. As a result, it has the same area as the FA of Fig. 4.

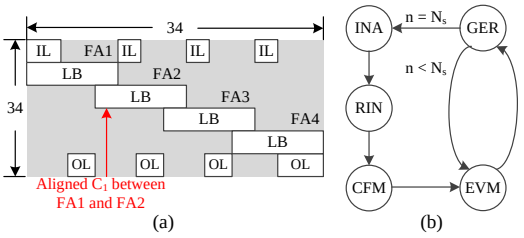


Fig. 12: Align Intermediate Signals: (a) Layout, (b) State Machine.

B. Scheme 2: Align Intermediate Signals

The area and execution steps of the four-bit adder in Fig. 11(b) can be further reduced by aligning the intermediate signals between computing elements. For instance, all the carries between FAs (e.g., FA1 and FA2) of Fig. 11(b) are aligned in the same column as shown in Fig. 12(a). Therefore, the columns initially used for carry transfer are removed. In addition, the carry of a FA can be directly stored in the minterms of the next stage FA, as the carry has been aligned in the same column. Hence, the extra rows initially allocated for interconnect, as well as the parts of IL and OL initially required to store carries, are removed. Finally, the four ILs to store the primary inputs are rearranged at the top of the crossbar, while the four OLs to store final results are rearranged at the bottom of the crossbar, as shown in Fig. 12(a). Fig. 12(b) shows the state machine required by the new implementation of Fig. 12(a). The adder receives all the primary inputs of the four FAs (i.e., C_0, A_i, B_i, $0 \leq i \leq 3$) at state RIN. Then, they are copied to the four FAs to configure their minterms at state CFM. For each stage, the state EVM generates the minterms of the corresponding FA, and state GER ANDs these minterms to generate the sum, carry, and their complements. It is worth noting that the number of execution steps are reduced from $7N_s$+1 (design of Fig. 7(a)) to $2N_s$+3; e.g., for N_s=4, the reduction is 62%. Moreover, the crossbar area is also reduced. For our 4-bit adder case study, the crossbar area is reduced from 46×40 (default design) to 34×34 (Fig. 12(a)); a reduction of 37%.

This optimization scheme is not applicable to all designs; it strongly depends on the place-and-route scheme. For instance, the design using the isolated scheme (e.g., Fig. 7(b)) does not support this scheme; the intermediate signals cannot be aligned in the same column since computing elements are isolated from each other.

C. Scheme 3: Combine Data Transfer and Inversion

Instead of producing intermediate result C_i and its complement $\overline{C_i}$ by each FA, we can rather produce only one of them (e.g., $\overline{C_i}$); the other one (e.g., C_i) will be generated while communicating the intermediate result to the next FA stage. Hence, the required crossbar column to produce C_i can be removed, resulting in less execution steps and area. Applying this scheme to default design of Fig. 7(b) results in the implementation shown in Fig. 13(a). The part where

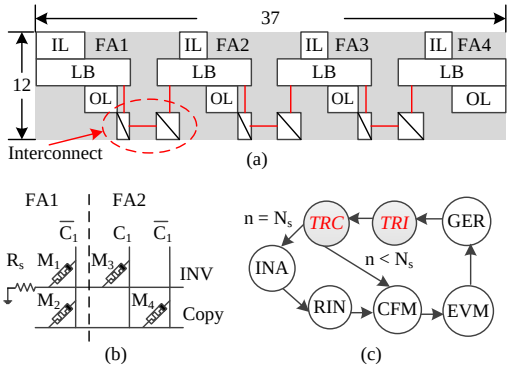


Fig. 13: Use Inversion to Transfer Data: (a) Layout, (b) Interconnect, (c) State Machine.

the combination of data transfer and inversion take place is highlighted, and illustrated in Fig. 13(b). The output $\overline{C_1}$ of FA1 is generated at state GER and directly stored in two memristors M_1 and M_2 of the interconnect. The interconnect feeds FA2 with C_1 and $\overline{C_1}$ via row INV and row Copy; C_1 is provided by inverting $\overline{C_1}$, while $\overline{C_1}$ is provided by just copying M_2 to M_4. In addition, the four ILs to store the primary inputs are rearranged at the top of the crossbar, while the four OLs to store final results are rearranged below the FAs as similar to Fig. 12(a).

Fig. 13(c) shows the state machine required by this new implementation; it makes use of two additional states; transfer using inversion (TRI), and transfer using copy (TRC). The adder receives all the primary inputs of the four FAs at state RIN. For each stage, state CFM configures the minterms of the corresponding FA; state EVM generates the results of these minterms; state GER provides logic AND of these minterms to obtain the complementary carry (e.g., $\overline{C_1}$); state TRI transfers C_i to the next stage using inversion; state TRC feeds the next FA stage with $\overline{C_i}$. Note that state TRI and TRC cannot be combined, because inversion and copy require different control voltages to the column related to output carry (e.g., $\overline{C_1}$ of FA1); see also Fig. 3(e) and (f). It is worth noting that the number of execution steps is reduced from $7N_s$+1=29 (design of Fig. 7(b)) to $5N_s$+2=22 steps; a reduction of 25%. Meanwhile, the area is also reduced from 12×40 to 12×37; a reduction of 7.5%.

V. EVALUATION

To validate the proposed approach, we select four designs as a case study, and perform two experiments on all of them. These four designs are four-bit ripple carry adders with different place-and-route and optimization schemes; they are:

- ID design: **I**nitial design based on **D**iagonal place-and-route scheme as shown in Fig. 7(a).
- II design: **I**nitial design based on **I**solated place-and-route scheme as shown in Fig. 7(b).
- OD design: An **O**ptimized version of **ID** design which is shown in Fig. 12(a); it incorporates the optimization

TABLE III: Description of Benchmarks

Circuit	Function	No. Input	No. Output	No. LUT4
alu4	ALU	14	8	1522
apex2	Logic	39	3	1878
apex4	Logic	9	19	1262
des	Data Encryption	256	245	1591
ex5p	Logic	8	63	1064
misex3	Logic	14	14	1397
pdc	Logic	16	40	4575
seq	Logic	41	35	1750
spla	Logic	16	46	3690

scheme 1 (i.e., calculate all outputs simultaneously) and 2 (i.e., align intermediate signals). Note that it is not possible to incorporate all the three discussed optimization schemes in the ID design. E.g., as the use of optimized scheme 2 removes the interconnects between FAs, scheme 3 (which makes use of the interconnect) cannot be used. Based on the nature of the three schemes, incorporating scheme 1 and 2 is the best in order to optimize delay.

- OI design: An **O**ptimized version of **II** design which is shown in Fig. 13(a); it incorporates optimization scheme 1 and 3. Again, incorporating all the three schemes is not possible. Hence, scheme 1 and 3 are the best to use with II design in order to optimize the delay.

The three experiments performed are the following.

- Verification of the mapping methodology; in this experiment, exhaustive SPICE simulations of all the designs are performed in order to check the correct functionality of the generated designs with all the possible input combinations.
- Evaluation and comparison of the four designs; in this experiment, all the four-selected designs are evaluated and compared in terms of area, delay and power consumption.
- To further illustrate the generality of our mapping methodology, it is applied to nine benchmarks selected from MCNC20 benchmark suite [50]. Table III summarizes the functions of the benchmarks and their input number, output number and number of required 4-input LUTs. These benchmarks spread in a wide range of input number, output number and circuit size (quantified by the number of LUTs). Area and delay are used as performance metrics to compare the initial and optimized designs.

Next, we first briefly review the simulation platform, and the parameters and models used to evaluate performance metrics. Thereafter, we provide the results.

A. Simulation Setup and Performance Metrics

The simulation platform consists of a memristor crossbar, a CMOS controller, and voltage drivers. The memristor model, controller, and voltage drivers are described by Verilog-A, and the crossbar array by SPICE netlist. The behavioural function of each of four selected designs is first verified using HSPICE simulator. Thereafter, each initial design and its optimized version are evaluated and compared with each other in terms of area, delay and power consumption. Note that memristor models are used for SPICE simulations, while the controller and drivers are described by Verilog-A so that the entire design (consisting of both the crossbar and CMOS part) can be simulated and verified using HSPICE simulator. To estimate the performance of CMOS controller, we will use a Verilog version.

TABLE IV: Simulation Parameters

Parameter	Description	Value
	Technology	
	Memristor (TaO_x) [49,51,52]	
F (nm)	Feature size	90
T (nm)	Thickness of TaO_x	8
V_{th} (V)	Threshold voltage	1.5
R_L ($k\Omega$)	Low resistance	200
$\frac{R_H}{R_L}$	–	7k
$\frac{R_{off}}{R_H}$	–	50
A_m (μm^2)	Area of a memristor	0.0324
T_{sw}(ns)	Switching time (max of SET and RESET)	1.71
E_m	Endurance of a memristor (max switching number)	10^{12}
	Nanowire (Copper) [53]	
κ	Dielectric constant of interlayer spacing	3.9
ρ ($\mu\Omega$cm)	Resistivity of Copper	8
T_{nw}(nm)	Thickness of the nanowire ($= F$)	90
W_{nw}(nm)	Width of the nanowire ($= F$)	90
C_{nw} (fF/μm)	Capcitance in unit length	0.26
R_{nw} (Ω/μm)	Resistance in unit length	9.88
	CMOS	
	Use UMC 90nm Library @ 500MHz)	
	Design [29]	
N_R	No. of rows in crossbar	–
N_C	No. of columns in crossbar	–
N_{step}	No. of execution steps	–
$n_{a,xbar}$	No. of all active memristors in the whole crossbar	–
V_w (V)	Program voltage	2.1
V_h (V)	Half-select voltage	1.05
$\frac{R_s}{R_L}$	–	10

To evaluate the benchmarks in Table III, a Matlab script is developed to read, map and estimate the metrics under consideration both for initial and optimized design versions; the optimized version uses diagonal mapping scheme and the optimization scheme 2. The inputs of the Matlab script are files in Berkeley logic interchange format (BLIF) of each benchmark [50]; BLIF consists of the minterms of each 4-input LUT which can be directly mapped to crossbar using our methodology.

Table IV summarizes the used simulation parameters; they are classified into technology and design parameters. The technology parameters are taken from [49,51–53], and provide realistic values for memristor as well as nanowires used to build the crossbar. For CMOS controller and voltage drivers, the UMC 90nm library is used. On the other hand, the design related parameters consist of those specifying the design itself (e.g., N_{step}, which is different for different designs), and those which specifies the requirements for the correctness of the design operations (e.g., values of control voltages of the crossbar) taken from [29,54]. Four parameters (i.e., N_R, N_C, N_{Step}, and $n_{a,xbar}$) are design dependent; hence for different benchmarks, they will have different values.

Three metrics are used to evaluate the performance: area, delay, and power consumption, while considering the

crossbar, CMOS voltage drivers and CMOS controller. Next, we will show how these evaluation metrics are determined. An adder is used as an example to illustrate the evaluation model and other benchmarks use the similar model.

The area of a single adder (A_{adder}) is expressed in Eq.8.

$$A_{\text{adder}} = \max\{A_{\text{xbar}}, A_{\text{cmos}}\} \tag{8}$$

where A_{adder} is defined as the maximum of the two values A_{xbar} and A_{cmos}, where A_{xbar} gives the crossbar area and A_{cmos} the area of entire CMOS part; note that we select only the max of the two values as the crossbar is stacked on the top of CMOS part.

A_{xbar} is a product of (N_{R}+1) crossbar rows by (N_{C}+1) columns as expressed in Eq.9.

$$\begin{cases} A_{\text{xbar}} & = (N_{\text{R}}+1)\cdot(N_{\text{C}}+1)\cdot A_{\text{m}} \\ A_{\text{m}} & = 4F^2 \end{cases} \tag{9}$$

where '+1' is needed as the implementation requires the use of a resistive element R_s to perform the appropriate logic operations (see e.g., Fig. 3(a)). A_{m} gives the memristor area.

A_{cmos} consists of the area of all voltage drivers ($A_{\text{vd,all}}$) and that of the control state machine (A_{ctrl}) as expressed in Eq.10.

$$\begin{cases} A_{\text{cmos}} & = A_{\text{vd,all}} + A_{\text{ctrl}} \\ A_{\text{vd,all}} & = A_{\text{vd,row}} + A_{\text{vd,col}} \\ & = \sum_{i=1}^{N_{\text{R}}}\left(30n_{\text{a,i}}F^2\right) + \sum_{j=1}^{N_{\text{C}}}\left(30n_{\text{a,j}}F^2\right) \\ & = 30F^2\left(\sum_{i=1}^{N_{\text{R}}} n_{\text{a,i}} + \sum_{j=1}^{N_{\text{C}}} n_{\text{a,j}}\right) \\ & = 30F^2(n_{\text{a,xbar}} + n_{\text{a,xbar}}) \\ & = 60n_{\text{a,xbar}}F^2 \end{cases} \tag{10}$$

The value of $A_{\text{vd,all}}$ is derived from Eq.6, which expresses the area of a single voltage driver used to drive a row or column with n_a active memristors. $A_{\text{vd,all}}$ is the sum of all the voltage drivers used to drive both rows and columns as shown in Eq.10, where $n_{\text{a,xbar}}$ is the total number of all the active memristors in the crossbar. The value of A_{ctrl} is provided by Cadence RTL compiler.

The delay of a single adder (D_{adder}) is expressed in Eq.11.

$$\begin{cases} D_{\text{adder}} & = N_{\text{step}}\cdot D_{\text{step}} \\ D_{\text{step}} & = D_{\text{xbar}} + D_{\text{ctrl}} \end{cases} \tag{11}$$

where D_{adder} is the product of the execution step number N_{step} and the delay of a single step D_{step}. D_{step} is the sum of the crossbar delay D_{xbar} and that of the CMOS controller D_{ctrl}.

The value of D_{ctrl} is provided by Cadence RTL compiler. On the other hand, D_{xbar} consists of the memristor switching time

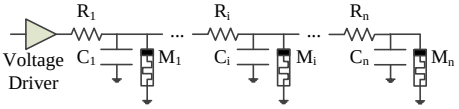


Fig. 14: Elmore RC Delay Model.

T_{sw} and the RC delay due to signal propagation through the nanowires D_{nw}; expressed in Eq.12.

$$\begin{cases} D_{\text{xbar}} & = T_{\text{sw}} + D_{\text{nw}} \\ D_{\text{nw}} & = \sum_{i=1}^{n}\left[C_i(\sum_{j=i}^{n} R_j)\right] \\ & = (n^2+4n-21/8)R_{\text{nw}}\cdot C_{\text{nw}}\cdot F^2 \text{ [55]} \\ n & = \max\{N_{\text{R}}, N_{\text{C}}\} \\ & \text{where} \\ R_1 & = \frac{3}{2}F\cdot R_{\text{nw}} \\ R_i & = 2F\cdot R_{\text{nw}}, \quad 1<i\leq n \\ C_1 & = \frac{3}{2}F\cdot\frac{C_{\text{nw}}}{2} \\ C_i & = 2F\cdot C_{\text{nw}}, \quad 1<i<n \\ C_n & = 2F\cdot C_{\text{nw}} + \frac{3}{2}F\cdot C_{\text{nw}} \end{cases} \tag{12}$$

where D_{nw} equals to the time to propagate the signal from the voltage driver to the nth memristor; it is given by Elmore model as shown in Eq.11 [53,55]. Fig. 14 shows the equivalent circuit to model the RC delay in a row or column of a crossbar. As the resistance value of the memristor device in its ON as well as in its OFF state is order of magnitudes higher than the nanowire resistance, the impact of memristor resistance in the modelled delay by Eq.11 is neglected [55]. Note that R_{nw} and C_{nw} denote the resistance and capacitance of nanowire in unit length (see Table IV).

The power consumed by a single adder P_{adder} is expressed by Eq.13, and is the sum of the power consumed by crossbar (P_{xbar}), by the CMOS voltage drivers ($P_{\text{vd,all}}$) and by the controller (P_{ctrl}) for all steps N_{step} to be executed.

$$\begin{cases} P_{\text{adder}} & = \sum_{n=1}^{N_{\text{step}}}(P_{\text{xbar}} + P_{\text{vd,all}} + P_{\text{ctrl}})_n \\ P_{\text{xbar}} & = \sum_{\text{all devices}} P_{\text{R}} \\ & = \sum_{\text{all devices}} \frac{V_{\text{R}}^2}{R} \end{cases} \tag{13}$$

For each execution step, P_{xbar} is the total power consumed by all the devices within the crossbar, which consists of the active and disabled memristors, and the resistors R_s. $P_{\text{R}}=\frac{V_{\text{R}}^2}{R}$ is the power consumed by each device, where V_{R} is the voltage across the device and R is its resistance, and they are both obtained using SPICE simulations. $P_{\text{vd,all}}$ is the power consumed by all the voltage drivers; we assume that the voltage drivers are almost ideal voltage sources and their power consumption is very small as compared with that consumed by the crossbar (which constitutes the load of the voltage drivers). The value of P_{ctrl} is provided by Cadence RTL compiler. To evaluate the power of each design, we first estimate the power for each input combination (2^8 in total), and thereafter calculate the average power consumption.

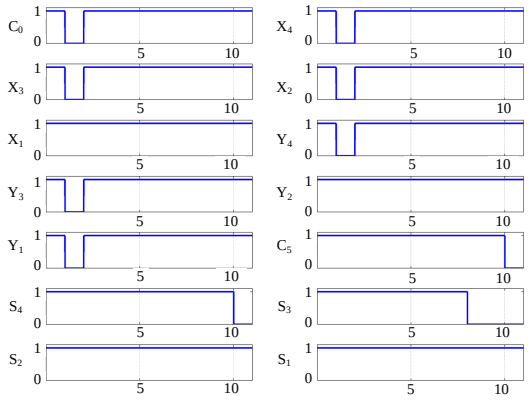


Fig. 15: Waveform of OD Adder SPICE Simulation.

B. Simulation Results

In the first experiment, all the four selected designs are exhaustively verified using SPICE simulations for all the possible input combinations. The simulation results have validated the correctness of the approach. For instance, Fig. 15 shows the waveform of a SPICE simulation of the OD design when input X='0001', Y='0010' and C_0=0. After eleven execution steps, the final results are C_5=0 and S='0011'. It is worth to note that the input data of the input latch are destructed after they are copied to the computing elements [29]; this has no impact on the correctness of the circuit as the inputs have been stored in minterms of the computing elements to calculate the final results.

In the second experiment, the performance of all the four selected designs is estimated in terms of area, delay and power consumption; the results are discussed next.

Area: Fig. 16(a) shows the area of the selected designs. Among all the designs, OI design uses the smallest crossbar as its FAs are adjacently placed. OD design requires the smallest CMOS part as its state machine is the simplest, and consumes the least overall area. The optimized designs require (up to 55%) less area than initial designs as their controllers have less states and less output control signals. Note that for these small designs, the CMOS area dominates the overall area. To further explore the scalability of the designs and their impact on area, we estimate the area of n-bit adders based on OD design (n=2^k, 2≤k≤7). Fig. 16(b) shows the area ratio of the crossbar over the CMOS part (including both the controller and voltage drivers). Clearly for larger adders (in this case for k>6), the crossbar area surpasses that of CMOS part.

Delay: Fig. 17(a) shows the required number of execution steps for each of the four designs, while Fig. 17(b) shows the corresponding delay for a single execution step D_{step} for each design and its breakdown; see also Eq.9. Obviously, the optimized designs have lower execution time and OD design performs by far the best. As each design is based on the same memristor crossbar technology, each design has the same

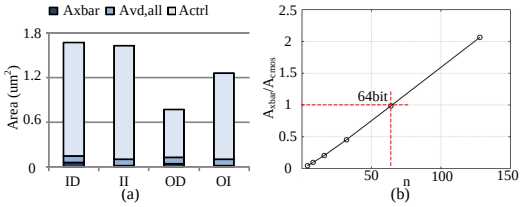


Fig. 16: Area of Selected Designs: (a) Area, (b) Scalability Exploration.

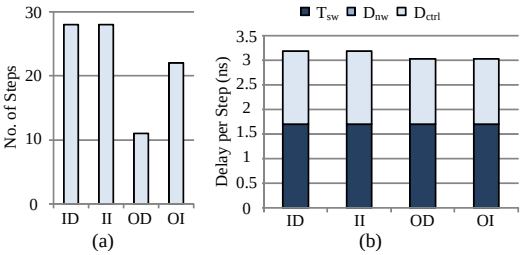


Fig. 17: Delay of Selected Designs: (a) Number of Execution Steps, (b) Delay per Execution Step.

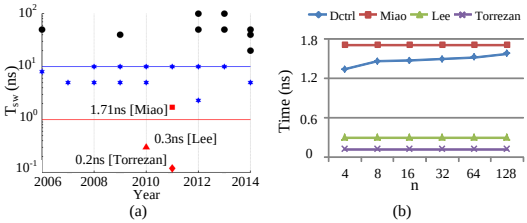


Fig. 18: Delay Scalability Exploration: (a) Tsw for Memristor Technology, (b) Dctrl and Tsw for n-Bit OD Designs.

memristor switching time T_{sw}=1.71 ns. The interconnect delay D_{nw} is negligible for all designs as compared with other delays. Moreover, CMOS controller delay D_{ctrl} for optimized designs is about 11% less than that of the initial designs.

The memristor technology is not mature yet and its switching time is still being improved. Fig. 18(a) shows the switching time of some reported resistive devices over the years [44,49,56–58], while Fig. 18(b) selects three reported T_{sw} and compares them with the controller delay for n-bit adders based on OD design (n=2^k, 2≤k≤7). Clearly, the CMOS delay may become the major critical component with respect to the performance of crossbar based logic designs. Nevertheless, the potential of the crossbar is enabling the massive parallelism, (where the same CMOS circuit may control different parallel designs within crossbar) and reducing the overall delay-operation of the whole design; hence this increases the overall throughput.

Power: Fig. 19 (a) shows the power consumption P_{adder} of a single adder and its breakdown for R_{L}=200kΩ [49], where

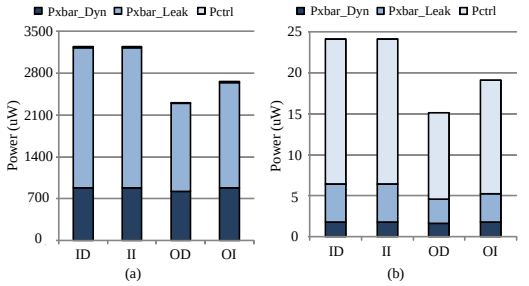


Fig. 19: Power Consumption of Selected Designs Using Different R_{L}: (a) R_{L}=200kΩ, (b) R_{L}=100MΩ.

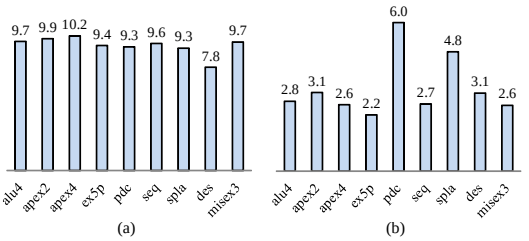


Fig. 20: Improvement Ratios for Different Benchmarks: (a) Area, (b) Delay.

R_{L} is the memristor resistance in its ON state; see also Eq.13. The power consumption of the crossbar P_{xbar} consists of the dynamic part ($P_{\mathrm{xbar,dyn}}$) as a result of resistive switching, and the leakage part ($P_{\mathrm{xbar,leak}}$) induced by sneak path currents. The difference between the dynamic power consumption of the four designs is marginal. However, the crossbar leakage is at least twice higher than the crossbar dynamic power. This highlights one of the major challenges the crossbar designs is facing, namely sneak path currents [52]; and shows the need of providing solutions. Note that the optimized designs consume about 30% less power than the initial designs, as they require less crossbar area. Fig. 19 also shows that the power consumption of the CMOS part P_{ctrl} is negligible.

One possible solution to reduce the power consumption in the crossbar (especially the part caused by sneak path currents) is to increase the value of R_{L}. For example, Fig. 19(b) shows the power consumption P_{adder} and its breakdown when R_{L} is increased to 100MΩ [59]. As a result, both the dynamic and leakage power consumption of the crossbar are significantly reduced, and the power consumed by CMOS controller becomes dominant now. Note that the optimized designs consume approximately 20% less power than the initial designs, as they have simpler CMOS controllers. Another possible solution to reduce the leakage power is reducing the duration of control voltages.

The results show systematically that the OD is the best design with respect to the three considered metrics.

In the third experiment, the performance of the nine benchmarks in Table III is estimated in terms of area and delay. Fig. 20 (a) and (b) shows the improvement ratios realized by optimized designs. As compared to the initial designs, the area of optimized designs is 7.8X to 10.2X smaller, and the delay is 2.2X to 6.0X shorter (due to a reduced number of execution steps).

Overall, the methodology can be used to not only map logic design on the crossbar, but also to evaluate its performance while considering optimization schemes. The results also show that the optimization techniques can significantly improve the performance of designs in terms of area and delay when the logic circuits become larger and more complex.

VI. CONCLUSION

This paper has proposed a mapping methodology of Boolean logic circuits on memristor crossbar as well as several optimization schemes. The performance of mapped logic circuits can be evaluated including both memristor and CMOS parts.

The proposed methodology provides the following advantages.

- **Generality**: The proposed methodology is potential to map arbitrary logic circuits as long as they are based on Boolean logic, such as adders, ALUs, data encryption (see also Table III).
- **Scalability**: The proposed methodology is scalable to map logic circuits as large as possible, but it should also consider the technology restrictions (e.g., sneak path current issues [43,44]) for appropriate functionality.
- **Automation**: The proposed methodology provides a potential to automate the mapping of large-scale logic circuits on memristor crossbar, which can be incorporated with existing logic synthesis tools (e.g., ODIN II [60] and ABC [61]).
- **Evaluation**: The proposed methodology provides performance evaluation for both the crossbar and CMOS part in terms of area, delay and power consumption, which can be used to compare between different designs based on resistive Boolean logic.
- **Modularity**: The proposed methodology uses a modular approach; this facilitates the improvement of the approach if need. E.g., in Fig. 5, the block 'implement functions by CEs' can be updated without the need to touch any other blocks in the flow.

In order to improve the logic based on memristor crossbar and related tools for mapping, more efforts should be paid to address the following challenges in the future work.

- **Support Other Logic Types**: As our method is modular, it is possible to be tuned to support other logic styles such as logic circuits proposed in [26,62–64].
- **Innovative Logic Design Styles**: As memristor logic circuit is still in infancy stage, innovative logic circuits based on resistive switching should be invented to maximize the potential of memristor crossbar. For instance, memristor crossbar may be suitable for analog circuits as a single memristor can represent a multi-level value, instead of a binary value [38].

- **Impact of Unreliable Technology**: Memristor technology is still under development, and therefore logic circuits based on crossbar are facing reliability challenges due to limited device endurance, device-to-device variation, cycle-to-cycle variation [44,65,66].
- **Sneak Path Issues**: The crossbar-based logic may fail due to the sneak path currents [34,35], which are the unexpected currents within the crossbar [43,44]. Even though several approaches have been proposed such as adding selector devices (e.g., CMOS transistor) [43,66], using complementary resistive switches [35], applying half-select voltages [34,38], how to efficiently apply these techniques to large-scale circuit is still under research. In addition, based on our simulations, when the ON/OFF current ratio increases, the impact of sneak path currents reduces.
- **Implementation Consideration**: Some effects should be considered in the future implementation. First, parasitic nanowire resistance causes the IR-drop. It can be solved by slightly increasing the control voltage V_w and V_h (e.g., 10%). In addition, increasing the low resistance is also helpful to reduce the impact of parasitic nanowire resistance. Second, the needed resistance ratio $\frac{R_H}{R_L}$ is typically 1000 to 10000. Third, the area of voltage driver is related to the placement of memristors within the crossbar. This may impact the scalability of the design. A possible solution is balancing the voltage drivers in the CMOS layer. Fourth, to isolate the crossbars, the nanowires should be broken and isolated materials (e.g., SiO_2) should be inserted. Some other possible solutions are reported in [67].
- **Complexity of CMOS Controller**: As the logic circuits based on memristor crossbar is scaling up, the complexity of CMOS controller is also increasing in order to compensate the driving force and support more execution states of the FSM. Therefore, it is crucial to design efficient CMOS control circuity in terms of area and delay being able to drive appropriate number of logic blocks in the crossbar. Some approaches may be helpful, such as sharing the CMOS controller between different logic circuits, pipelining the computing elements to simplify the controller, etc. In addition, a simpler controller is likely to have a shorter delay. Therefore, it is possible to reduce the power consumption due to a shorter duration of applied control voltages.

Overall, the proposed mapping methodology sets a step towards the implementation of large-scale resistive computing architectures, and provides an opportunity to examine the potential of memristor crossbar architectures.

REFERENCES

[1] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.

[2] S. Borkar, "Design perspectives on 22nm cmos and beyond," in *Proceedings of the 46th Annual Design Automation Conference (DAC)*. ACM, 2009, pp. 93–94.

[3] S. Hamdioui *et al.*, "Reliability challenges of real-time systems in forthcoming technology nodes," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. EDA Consortium, 2013, pp. 129–134.

[4] J. L. Hennessy *et al.*, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[5] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture (MICRO)*. IEEE, 1999, p. 2.

[6] D. E. Nikonov *et al.*, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, vol. 101, pp. 2498–2533, 2013.

[7] A. Bachtold *et al.*, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, pp. 1317–1320, 2001.

[8] L. Ding *et al.*, "Cmos-based carbon nanotube pass-transistor logic integrated circuits," *Nature communications*, vol. 3, p. 677, 2012.

[9] L. Amarú *et al.*, "New logic synthesis as nanotechnology enabler," *Proceedings of the IEEE*, vol. 103, pp. 2168–2195, 2015.

[10] B. Behin-Aein *et al.*, "Proposal for an all-spin logic device with built-in memory," *Nature nanotechnology*, vol. 5, pp. 266–270, 2010.

[11] B. Behin-Aein *et al.*, "All-spin logic," in *Device Research Conference (DRC), 2010*. IEEE, 2010, pp. 41–42.

[12] M. Fuhrer *et al.*, "Spintronics: A path to spin logic," *Nature physics*, vol. 1, pp. 85–86, 2005.

[13] L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507–519, 1971.

[14] D. B. Strukov *et al.*, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008.

[15] ITRS, "Beyond cmos white paper," 2014.

[16] W. Zhao *et al.*, "Nanodevice-based novel computing paradigms and the neuromorphic approach," in *Circuits and Systems (ISCAS), IEEE International Symposium on*. IEEE, 2012, pp. 2509–2512.

[17] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the Design, Automation & Test in Europe (DATE)*. EDA Consortium, 2015, pp. 1718–1725.

[18] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Proceedings of the Design, Automation & Test in Europe (DATE)*. EDA Consortium, 2017, pp. 1729–1725.

[19] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 15)*. IEEE, 2015, pp. 57–62.

[20] A. Haron *et al.*, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[21] J. Yu *et al.*, "Skeleton-based design and simulation flow for computation-in-memory architectures," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*. IEEE, 2016, pp. 165–170.

[22] L. Yavits *et al.*, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, pp. 148–151, 2015.

[23] S. Li *et al.*, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 173.

[24] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 115–119, 2013.

[25] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.

[26] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.

[27] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[28] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.

[29] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[30] Intel xpoint memory. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/non-volatile-memory.html

[31] Crossbar 3d rram. [Online]. Available: http://www.crossbar-inc.com/

[32] A. Raghuvanshi *et al.*, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 470–477.

[33] F. S. Marranghello *et al.*, "Sop based logic synthesis for memristive imply stateful logic," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 228–235.

[34] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 2054–2066, 2014.

[35] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 64–74, 2015.

[36] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 211–221, 2013.

[37] X. Guan *et al.*, "A spice compact model of metal oxide resistive switching memory with variations," *IEEE electron device letters*, vol. 33, pp. 1405–1407, 2012.

[38] K.-H. Kim *et al.*, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, pp. 389–395, 2011.

[39] X. Zhu *et al.*, "Performing stateful logic on memristor memory," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 60, pp. 682–686, 2013.

[40] B. Parhami, *Computer arithmetic*. Oxford university press, 1999, vol. 20, no. 00.

[41] P. C. McGeer *et al.*, "Espresso-signature: A new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, pp. 432–440, 1993.

[42] L. Xie *et al.*, "Interconnect networks for memristor crossbar," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 124–129.

[43] S. Hamdioui *et al.*, "Memristor based memories: Technology, design and test," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 9th IEEE International Conference On*. IEEE, 2014, pp. 1–7.

[44] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.

[45] X. Tang *et al.*, "A high-performance low-power near-vt rram-based fpga," in *Field-Programmable Technology (FPT), 2014 International Conference on*. IEEE, 2014, pp. 207–214.

[46] P.-E. Gaillardon *et al.*, "Design and architectural assessment of 3-d resistive memory technologies in fpgas," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 40–50, 2013.

[47] M. Mao *et al.*, "Optimizing latency, energy, and reliability of 1t1r reram through appropriate voltage settings," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 359–366.

[48] J. M. Rabaey *et al.*, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.

[49] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Advanced materials*, vol. 23, pp. 5633–5640, 2011.

[50] "Fpga place-and-route challenge." [Online]. Available: http://www.eecg.toronto.edu/ vaughn/challenge/challenge.html

[51] M. Zangeneh *et al.*, "Performance and energy models for memristor-based 1t1r rram cell," in *Proceedings of the great lakes symposium on VLSI*. ACM, 2012, pp. 9–14.

[52] J. J. Yang *et al.*, "The mechanism of electroforming of metal oxide memristive switches," *Nanotechnology*, vol. 20, p. 215201, 2009.

[53] D. B. Strukov *et al.*, "Cmol fpga: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, p. 888, 2005.

[54] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2016 International Conference on*. IEEE, 2016, pp. 1–6.

[55] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, pp. 55–63, 1948.

[56] M.-J. Lee *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5- x/tao2- x bilayer structures," *Nature materials*, vol. 10, pp. 625–630, 2011.

[57] A. C. Torrezan *et al.*, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, p. 485203, 2011.

[58] Crossbar 3d rram. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends

[59] C. Schindler *et al.*, "Electrode kinetics of cu-sio2-based resistive switching cells: Overcoming the voltage-time dilemma of electrochemical metallization memories," *Applied physics letters*, vol. 94, p. 2109, 2009.

[60] P. Jamieson *et al.*, "Odin ii-an open-source verilog hdl synthesis tool for cad research," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*. IEEE, 2010, pp. 149–156.

[61] Abc: A system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/abc.htm

[62] E. Linn *et al.*, "Beyond von neumann? logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, p. 305205, 2012.

[63] S. Kvatinsky *et al.*, "Mrlmemristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012, pp. 1–6.

[64] S. Kvatinsky *et al.*, "Magicmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.

[65] H.-S. P. Wong *et al.*, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, 2012.

[66] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, pp. 2632–2663, 2009.

[67] I. Vourkas *et al.*, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 206–217, 2016.

Lei Xie (S'15) received his Bachelors and Masters degree in Microelectronics from Xian Jiaotong University, Xian, China. Currently, he is pursuing a PhD at the Computer Engineering Lab in the Delft University of Technology, Delft, the Netherlands. His research interest is memristor-based logic circuit design.

Hoang Anh Du Nguyen (S'15) received the M.Sc. degrees in computer engineering from the Delft University of Technology, Delft, The Netherlands. She is currently a pursuing the Ph.D. degree with the Computer Engineering Laboratory, Delft University of Technology. Her current research interests include Computation-in-Memory (CIM) architecture for Big-Data, memristor based systems and synthesis automation.

Mottaqiallah Taouil (S'10 – M'15) received the M.Sc. and Ph.D. degrees (both with Hons.) in computer engineering from the Delft University of Technology, Delft, The Netherlands. He is currently a Post-Doctoral Researcher with the Dependable Nano-Computing Group, Delft University of Technology. His current research interests include reconfigurable computing, embedded systems, very large scale integration design and test, built-in-self-test, and 3-D stacked integrated circuits, architectures, design for testability, yield analysis, and memory test structures.

Said Hamdioui (M'99 – SM'11) is currently a Chair Professor on Dependable and Emerging Computer Technologies at the Computer Engineering Laboratory of the Delft University of Technology (TUDelft), the Netherlands. Prior to joining TUDelft, Hamdioui worked for Intel Corporation (Califorina, USA), Philips Semiconductors R&D (Crolles, France) and for Philips/ NXP Semiconductors (Nijmegen, The Netherlands). His research focuses on two domains: Dependable CMOS nano-computing (including Reliability, Testability, Hardware Security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, in-memory-computing). He owns one patent and has published one book and co-authored over 170 conference and journal papers. He delivered dozens of keynote speeches, distinguished lectures, and invited presentations and tutorial at major international forums/conferences/schools and at leading semiconductor companies. Hamdioui is a Senior member of the IEEE, Associate Editor of IEEE Transactions on VLSI Systems (TVLSI), and he serves on the editorial board of IEEE Design & Test, and of the Journal of Electronic Testing: Theory and Applications (JETTA). He is also member of AENEAS/ENIAC Scientific Committee Council (AENEAS =Association for European NanoElectronics Activities).

Koen Bertels is Professor and Head of the Computer Engineering Laboratory at Delft University of Technology. His research focuses on heterogeneous multicore computing, investigating topics ranging from compiler technology, runtime support and architecture. He recently started working on quantum computing as a principal investigator in the Qutech research center. He served as general and program chair for various conferences such as FPL, RAW, ARC. He co-authored more than 30 journal papers and more than 150 conference papers.

Non-Volatile Look-up Table Based FPGA Implementations

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels, Mohammad Alfailakawi*
Laboratory of Computer Engineering, Delft University of Technology, the Netherlands
*Computer Engineering Department, University of Kuwait, Kuwait
Email: {L.Xie,H.A.DuNguyen,M.Taouil,S.Hamdioui,K.L.M.Bertels}@tudelft.nl;alfailakawi.m@ku.edu.kw

***Abstract*—Many emerging technologies are under investigation to realize alternatives for future scalable electronics. Memristor is one of the most promising candidates due to memrsitor's non-volatility, high integration density, near-zero standby power consumption, etc. Memristors have been recently utilized in non-volatile memory, neuromorphic system, resistive computing architecture, and FPGA to name but a few. An FPGA typically consists of configurable logic blocks (CLBs), programmable interconnects, configuration, and block memories. Most of the recent work done was focused on using memristor to build FPGA interconnects and memories. This paper proposes two novel FPGA implementations that utilize memristor-based CLBs and their corresponding automatic design flow. To illustrate the potential of the proposed implementations, they are benchmarked using Toronto 20, and compared with the state-of-the-art in terms of area and delay. The experimental results show that both the area (up to 4.24x) and delay (up to 1.46x) of the novel FPGAs are very promising.**

5

I. INTRODUCTION

As transistors gradually approach their inherent physical device limits, CMOS technology faces major challenges such as increased leakage, saturated performance gain, and reduced reliability [1,2]. To address these challenges, novel technologies; such as carbon nanotube, graphene transistors, and memristors [3]; are proposed as alternatives for future scalable electronics. Memristors are one of the most promising candidates due to their non-volatility, high integration density, and near-zero standby power [3,4]. Memristors-based design have been proposed for non-volatile memory [4], neuromorphic system, resistive computing architecture [5,6], and field programmable gate array (FPGA) [7–9].

Many novel memristor-based FPGAs, called MemFPGA for short, have been reported recently. MemFPGAs typically employ the classical island-style architecture [10] which consists of configurable logic blocks (CLBs), programmable interconnect, and block RAM (BRAM). Each CLB consists of look-up tables (LUTs) and a D flip-flop (DFF). Both CLBs and the programmable interconnect use memories to store configuration information. In MemFPGAs, memristors are utilized in the following fashion:

- As configuration memory for CLBs and interconnects [7]
- Used in the implementation of programmable interconnect [8].
- Implement BRAMs and DFFs [9].

In this work, we propose the use of memristors to improve the implementation of LUTs within FPGA, something that have not been done before. This paper proposes two FPGA implementations using memristor-based LUTs along with an appropriate Electronic Design Automation (EDA) process. The cost of both implementations in terms of area and delay is analyzed.

The rest of this paper is organized as follows. Section II briefly describes memristor-based logic. Section III presents the proposed FPGA implementation followed by an EDA flow in Section IV. Section V evaluates the proposed approaches and the paper is concluded in Section IV.

II. FUNDAMENTALS OF MEMRISTOR LOGIC

This section starts with the characteristics of memristors, followed by an overview of memristor logic, finally presents two memristor logic styles used in this work.

A. Electronic Characteristics of Memristor

Fig. 1 shows I-V characteristics of a typical memristor [4]. The memristor switches from one resistive state to another when voltage across the device is greater than its threshold voltage V_{th}. Otherwise, it stays in its current resistive state. The the high-to-low switching is referred to as *SET* (R_L) resistance, while low-to-high switching is referred to as *RESET*(R_L).

B. Overview of Memristor Logic

There are four types of memristor logic that have been previously proposed, namely, threshold [11], majority [11], implication [12], and Boolean logic [13,14]. Since LUTs are commonly based on Boolean logic, we will limit our discussion to memristor-based Boolean logic. Memristor-based Boolean logic can be classified into two styles depending on how logic states are represented. One style uses high and low *voltages* to represent logic 1 and 0 as is referred to as memristor-ratioed logic (MRL) [13]. On the other hand, when

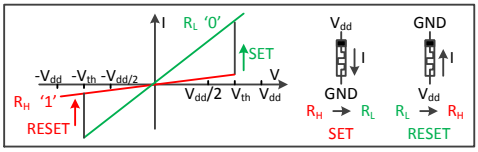


Fig. 1: Electronic Characteristics of Memristor

978-1-5090-4900-4/16/$31.00 © 2016 IEEE

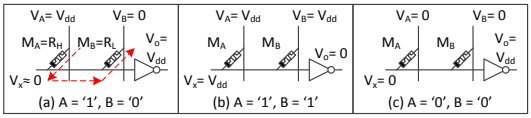


Fig. 2: 2-Input NAND Gate of Memristor-Ratioed Logic

high/low *resistance* is to represent logic 1 and 0, then it is referred to as Resistive Boolean logic (RBL) [14]. Next, we describe MRL and RBL Boolean logic.

C. Memristor Ratioed Logic (MRL)

The basic gates in MRL are AND, OR, NAND and NOR [13]. Fig. 2 shows an example of a 2-input NAND gate consisting two memristors and a CMOS inverter (an *n*-input NAND gate requires *n* memristors). When only one the inputs is 1 (e.g., A=1, B=0, see Fig. 2(a)), a current flows through memristor M_A and M_B as indicated by the dash-lined resulting in RESETing M_A and SETing M_B. The voltage of the floating nanowire $V_x = \frac{M_B}{M_A+M_B} V_{dd} \approx 0$ as $R_H \gg R_L$ [4], hence the output voltage V_o is V_{dd}. Cases when both inputs are 1 or 0 can be analyzed similarly and are shown in Fig. 2(b)-(c).

D. Resistive Boolean Logic (RBL)

In RBL, the basic logic gates are NAND, copy, invert (INV), and AND and are shown in Fig. 3 [14]. To illustrate working principle of RBL, a two-input NAND gate is used as an example. A 2-input NAND gate consists of two input memristors (M_A and M_B), an output memristor (M_o), and a resistor R_s ($R_L \ll R_s \ll R_H$). The output memristor must be RESET to R_H before each operation and input ones must be pre-programmed before execution (for brevity, this initialization is not shown). To perform an NAND operation, control voltages $\frac{V_{dd}}{2}$ and V_{dd} ($\frac{V_{dd}}{2} < V_{th}$) are applied to input and output memristors, respectively. In the case when input A=1 and B=0, $M_A = R_H$, and $M_B = R_L$ (see Fig. 3(a)), the voltage on floating nanowire $V_x \approx \frac{V_{dd}}{2}$ is $R_L \ll R_s \ll R_H$ resulting in the voltage across M_o to be $V_{mo} \approx V_{dd} - \frac{V_{dd}}{2} = \frac{V_{dd}}{2} < V_{th}$, rendering M_o to stay in R_H state. Case when input A=B=1 can be analyzed similarly and is shown at the bottom of Fig. 3(a). Fig. 3(b) shows other gates which work in the similar way as the 2-input NAND gate.

III. Two FPGAs Using Memristor Logic

This section first briefly describes island-style FPGA architecture then presents MRL and RBL based FPGA architectures.

A. Island-Style FPGA Architecture

Fig. 4(a) shows the island-style FPGA architecture [10] which consists of CLBs, connection boxes (CBs), switching boxes (SBs) and BRAMs. Each CLB is composed of a switch matrix and *N* basic logic elements (BLEs) as shown in Fig. 4(b). The switch matrix contains multiple MUXs configured by SRAM bits to route *I* shared inputs and feedback *N* outputs among BLEs. A single *K*-input BLE contains a LUT, DFF, MUX and configuration memories (i.e., SRAM) as shown in Fig. 4(c).

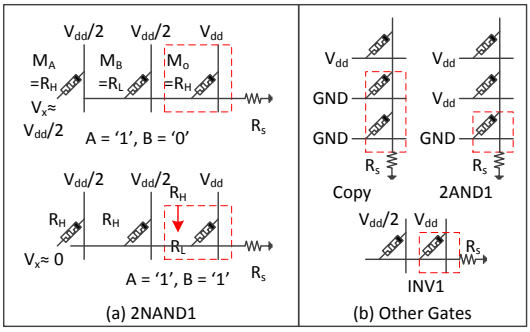


Fig. 3: Logic Gates of Resistive Boolean Logic

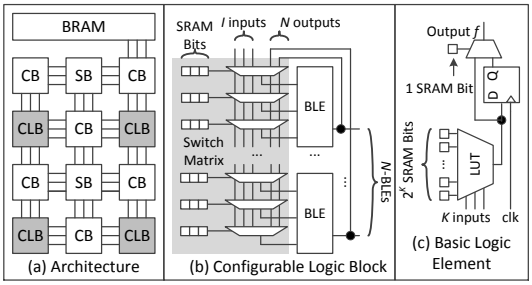


Fig. 4: Island-Style FPGA Architecture

The LUT can implement a *K*-input Boolean function as determined by the configuration memories. The LUT can switch between sequential and combinational mode using the DFF and MUX. CBs and SBs constitute the programmable interconnect to route the signals among CLBs.

The reminder of this section presents two novel FPGAs. The MRL based implementation will be referred to as ***MFPGA*** whereas RBL based one will be labeled as ***RFPGA***.

B. MFPGA

MFPGA uses MRL to implement the LUT and switch matrix (SM) of the CLBs while still using CMOS to implement the DFF and 2:1 MUX of BLEs. The output f of a 2-input LUT can be expressed by Eq.1:

$$f = c_1\bar{x_1}\bar{x_2} + c_2\bar{x_1}x_2 + c_3x_1\bar{x_2} + c_4x_1x_2 \quad (1)$$
$$= \overline{c_1\bar{x_1}\bar{x_2}} \cdot \overline{c_2\bar{x_1}x_2} \cdot \overline{c_3x_1\bar{x_2}} \cdot \overline{c_4x_1x_2}$$

where x_i (i=1,2) are the inputs and c_i ($1 \le i \le 4$) are configuration bits. Fig. 5(a) shows an example of an MRL-based 2-input LUT. Each term in output f, e.g., $\overline{c_1\bar{x_1}\bar{x_2}}$, is implemented using a NAND gate whose output are used as inputs to another NAND gate to calculate the complete output f. All memristors of the NAND gates are mapped on a memristor crossbar. For instance, the term $\overline{c_1\bar{x_1}\bar{x_2}}$ is realized by enabling the three memristors at the junctions between columns $\bar{x_1}$, $\bar{x_2}$,

5

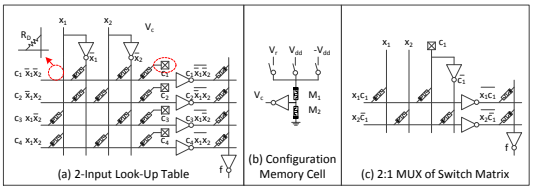


Fig. 5: MFPGA

and c_1 with the first row while keeping other memristors on the same row disabled (by not electroforming them [4]). Note that the disabled junction is permanently in a high resistance $R_D \gg R_H$ [14]. Configuration bit c_1 is stored in a 3-Transistor-2-Memristor (3T2R) cell as shown in Fig. 5(b) [7]. Two memristors M_1 and M_2 form a voltage divider and inverter output V_c is used as value for c_1. To configure c_1 to 0, M_1 and M_2 should be programmed to R_{L} and R_{H} using V_{dd}. During execution stage, V_{f} is applied resulting in V_c=0 outputted. The case of c_1=1 works in a similar fashion.

The output f of an N:1 MUX (e.g., N=2) used in SM can be expressed by Eq.2 where x_i (1≤i≤2) presents the inputs and c_1 configuration bit. Fig. 5(c) shows an MRL-based 2:1 MUX as an example. It works similarly as MRL LUT.

$$f = x_1 c_1 + x_2 \bar{c_1} = \overline{\overline{x_1 c_1} \cdot \overline{x_2 \bar{c_1}}} \tag{2}$$

C. RFPGA

In RFPGA, RBL is used to implement BLEs while it uses the same switch matrix used in MFPGA. The output f of a K-input LUT can be expressed by Eq.1; e.g., K=2.

$$f = \overline{c_1 \bar{x_1} \bar{x_2}} \cdot \overline{c_2 \bar{x_1} x_2} \cdot \overline{c_3 x_1 \bar{x_2}} \cdot \overline{c_4 x_1 x_2} \tag{3}$$

Fig. 6(a) shows an RBL implementation of 2-input LUT. The first two rows are used to invert x_i to $\bar{x_i}$ (i=1,2) whereas the following four rows implement the four terms in Eq.3 by mapping four NAND gates of Fig. 3(a) on the crossbar. For instance, the expression $\overline{c_1 \bar{x_1} \bar{x_2}}$ is implemented by row 3 where three memristors are placed at columns $\bar{x_1}$, $\bar{x_2}$ and $\bar{c_1}$ junctions representing inputs while a memristor is placed on column f junction signifying the output. The remaining junctions in row 3 are disabled. To calculate output f, an AND gate is mapped on column f where it reuses the output of the four NAND gates as input hence storing value at the memristor at the junction of row and column f.

To control the crossbar of the BLE, multiple voltage drivers and a CMOS controller are employed. A voltage driver is attached to each nanowire (triangles in Fig. 6(a)) and is used to control the voltage on the nanowires. The controller has two modes of operation, configuration (CFG) and execution (EXE) as described by the state machine shown in Fig. 6(b). The CFG mode consists of two states:

1) RSC: Activate all configuration bits by RESET all memristors to R_H.
2) WR0: Deactivate configuration bits that do not contribute to function implemented (set memristors to R_L).

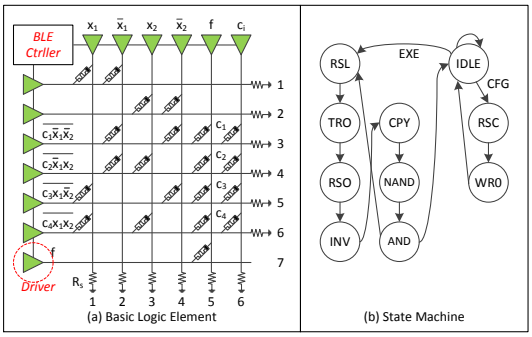


Fig. 6: RFPGA

TABLE I: Control Voltages for BLEs of RFPGA

EXE Mode							
State	Row			Column			
	INV	NAND	OL	IN	INN	C	OUT
	1–2	3–6	7	x_1,x_2	$\bar{x_1}$,$\bar{x_2}$	c_i	f
RSL	V_{dd}	V_{dd}	$V_{dd}/2$	*GND*	*GND*	$V_{dd}/2$	*GND*
TRO	*GND*	$V_{dd}/2$	V_{dd}	Float	$V_{dd}/2$	$V_{dd}/2$	Float
RSO	$V_{dd}/2$	$V_{dd}/2$	V_{dd}	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	*GND*
INV	Float	$V_{dd}/2$	$V_{dd}/2$	V_{dd}	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$
CPY	V_{dd}	*GND*	$V_{dd}/2$	Float	Float	$V_{dd}/2$	$V_{dd}/2$
NAND	$V_{dd}/2$	Float	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	V_{dd}
AND	$V_{dd}/2$	V_{dd}	*GND*	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	Float
CFG Mode							
State	Row			Column			
	INV	NAND	OL	IN	INN	C	OUT
RSC	$V_{dd}/2$	V_{dd}	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	*GND*	$V_{dd}/2$
WR0	$V_{dd}/2$	V_{dd}/*GND**	$V_{dd}/2$	$V_{dd}/2$	$V_{dd}/2$	V_{dd}	$V_{dd}/2$

The EXE mode consists of seven states:

1) RSL: RESET all memristors to R_H except output f and configuration bits (i.e., c_i, 1≤i≤4).
2) TRO: Transfer output f to the next BLEs, while receive all inputs of the LUT.
3) RSO: RESET the memristor that stores output f.
4) INV: Invert inputs x_i to $\bar{x_i}$ (i=1,2).
5) CPY: Copy inputs to all NAND gates.
6) NAND: Execute all NAND gates to calculate the items.
7) AND: Execute an AND gate to calculate output f.

To perform the operation of each state in the state machine, control voltages as indicated in Table I needs to be applied to the various nanowires. For instance, during CPY state, all inputs (x_i and $\bar{x_i}$) are copied to all NAND gates by applying V_{dd} to row INV (row 1–2) and *GND* to NAND (row 3–6) while simultaneously column IN (x_i) and INN $\bar{x_i}$ are floating. In order to reduce the impact of *sneak path currents* on the BLE's robustness, $\frac{V_{\mathrm{dd}}}{2}$ is applied to rows and columns that are not involved in the operations [4,14].

IV. AN EDA FLOW FOR PROPOSED FPGAS

This section presents a modified EDA flow for the proposed FPGA architectures and evaluate them in term of area and delay.

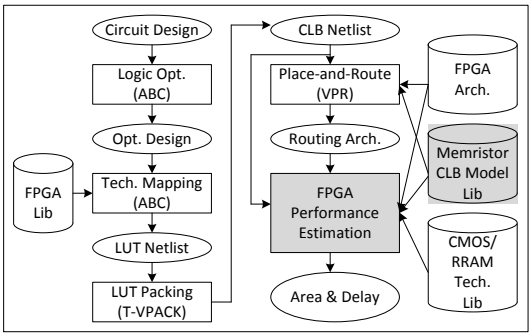


Fig. 7: A CAD Flow for MFPGA and RFPGA

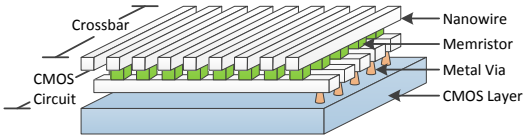


Fig. 8: A Possible Implementation of MFPGA and RFPGA

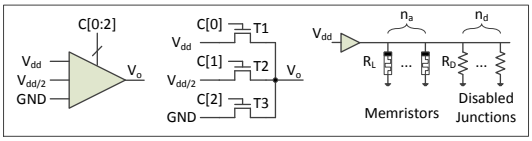


Fig. 9: CMOS Voltage Drivers.

A. Modified EDA Flow

To automatically implement circuits using the proposed FPGAs, we modify the standard EDA flow for CMOS FPGA [10] as shown in Fig. 7 where shaded blocks identify modification. Circuit design is first optimized at logic level and then mapped to LUTs using ABC [15]. Thereafter, LUTs are packed into CLBs using T-VPACK [16] where each CLB consists of one or more LUTs depending on configuration setting. Next, CLB netlist is placed and routed using VPR [16]. Finally, the entire FPGA consisting of CLBs and the routing architecture (including CBs and SBs) are estimated in terms of area and delay. To estimate the performance of the proposed architectures, we modify the performance estimation block by adding area and delay models for memristor-based CLBs. As the FPGA architecture is regular, FPGA performance estimation block only need to sum up the area and delay of CLBs and routing architecture [10]. The rest of this section presents area and delay model for the proposed CLBs.

B. Area Model of Memristor CLBs

CLBs of MFPGA and RFPGA can be implemented by a memristor crossbar stacked on top of a CMOS circuit as shown in Fig. 8 [3,4]. Hence, the area of a single CLB (A_{clb}) is estimated by the maximum area of the memristor crossbar (A_{xbar}) and CMOS circuit (A_{cmos}) as given by Eq.4. Crossbar area is estimated as the product of number of junctions (N_{junction}) by the area of a single junction (A_{junction}).

$$\begin{cases} A_{\text{clb}} &= \max\{A_{\text{xbar}}, A_{\text{cmos}}\} \\ A_{\text{xbar}} &= N_{\text{junction}} \cdot A_{\text{junction}} \end{cases} \tag{4}$$

The CMOS part of MFPGA CLB contains inverters, DFFs, and 3T2R memory cells, hence can be represented as the summation of each component's area as expressed in Eq.5.

$$A_{\text{cmos,MFPGA}} = \sum A_{\text{inv}} + \sum A_{\text{memory}} + \sum A_{\text{dff}} \tag{5}$$

where A_{inv}, A_{memory}, and A_{dff} are the areas of a CMOS inverter, 3T2R memory cell, a DFF, respectively.

The CMOS part of RFPGA CLB contains voltage drivers and controller and thus its area can be expressed in Eq.6.

$$A_{\text{cmos,RFPGA}} = A_{\text{ctrl}} + \sum A_{\text{vd}} \tag{6}$$

where A_{ctrl} presents the area of the controller and A_{vd} is that of a single voltage driver. The area of the controller can be estimated by a synthesis tool (e.g., Cadence RTL compiler).

Fig. 9 shows a possible implementation of a voltage driver consisting of three pass transistors [17] controlled by three-bit signals C[0:2]. To drive a nanowire with multiple memristors connected as shown on the right side of Fig. 9, the transistors should provide enough current to drive such wires. Therefore, transistors width-to-length ratio $\frac{W}{L}$ should be carefully determined. To program a *single* memristor, the transistor must supply a current greater than $I_{\text{w}}=\frac{V_{\text{dd}}}{R_L}$ [9]. The area of a transistor is typically $A_{\text{n}}=6F^2$ [7] where F is the feature size of CMOS technology. To drive n_a active memristors in parallel, $\frac{W}{L}$ should be increased n_a times in order to provide the required current I_{w}. As a result, A_{n} of the transistor increases n_a times as given in Eq. 7.

$$I_{\text{w}} = n_a \frac{V_{\text{dd}}}{R_L}, \text{and } A_{\text{n}} = 6n_aF^2 \tag{7}$$

Further, assume that we have another n_d disabled junctions with each consuming current equal to $I_{\text{D}}=\frac{V_{\text{dd}}}{R_D}$, then the transistor should also compensate for the current through n_d disabled memristors as described in Eq.8.

$$\begin{cases} I_{\text{w}} = n_a\frac{V_{\text{dd}}}{R_L} + n_d\frac{V_{\text{dd}}}{R_D} = (n_a + \frac{R_L}{R_D}n_d)\frac{V_{\text{dd}}}{R_L}, \\ A_{\text{n}} = 6(n_a + \frac{R_L}{R_D}n_d)F^2 \end{cases} \tag{8}$$

Finally, the total area A_{vd} of a single voltage driver is as given in Eq.9. Typically, $\frac{R_D}{R_L}>5\times10^4$ [18] and hence the number of memristors n_a dominates the area of the driver.

$$A_{\text{vd}} = 3A_{\text{n}} = 18(n_a + \frac{R_L}{R_D}n_d)F^2 \approx 18n_aF^2 \tag{9}$$

5

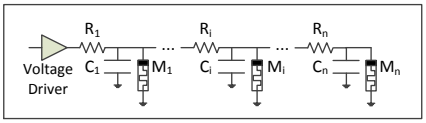


Fig. 10: Elmore Delay Model

C. Delay Model of Memristor CLBs

The delay of MFPGA CLB is determined by the critical path from inputs to outputs similar to CMOS circuit. The critical path contains an SM MUX , an LUT, a DFF, and a 2:1 MUX. The DFF and 2:1 MUX are implemented using CMOS and the remaining components are implemented using MRL. The delay D_{mrl} of a MRL-based LUT or MUX is modelled by Eq.10 (see Fig. 5).

$$D_{\text{mrl}} = 2D_{\text{inv}} + 2T_{\text{sw}} + D_{\text{nw,row}} + D_{\text{nw,col.}} \tag{10}$$

where D_{inv} represents the delay of a CMOS inverter, T_{sw} is the switching time of a memristor device, and $D_{\text{nw,row}}$ ($D_{\text{nw,col.}}$) the delay of a row (column) nanowire. Note that memristors driven by V_{dd} first switch R_{L} and then memristors driven by *GND* switch to R_{H} or vice versa (see Fig. 2(a)) [13], therefore, memristor devices need $2T_{\text{sw}}$ to switch. A row or column nanowire is modelled as a transmission line as shown in Fig. 10 and its delay D_{nw} is formulated by Elmore model [19] as given in Eq.11.

$$\begin{cases} D_{\text{nw}} & = \sum_{i=1}^{n} \left[C_i \left(\sum_{j=1}^{i} R_j \right) \right] \\ & = (n^2 + 4n - 21/8) R_{\text{nw}} \cdot C_{\text{nw}} \cdot F^2 \\ n & : \text{number of junctions in the nanowire} \\ R_1 & = \frac{3}{2} F \cdot R_{\text{nw}} \\ R_i & = 2F \cdot R_{\text{nw}}, \quad 1 < i \leq n \\ C_1 & = \frac{3}{2} F \cdot \frac{C_{\text{nw}}}{2} \\ C_i & = 2F \cdot C_{\text{nw}}, \quad 1 < i < n \\ C_n & = 2F \cdot C_{\text{nw}} + \frac{3}{2} F \cdot C_{\text{nw}} \end{cases} \tag{11}$$

RFPGA CLB contains several BLEs based on RBL and an SM based on MRL. The delay of SM is modelled by Eq.10. The delay D_{ble} of the BLE of Fig. 6 is modelled by Eq.12.

$$\begin{cases} D_{\text{ble}} & = N_{\text{step}} \cdot D_{\text{step}} = 7 \cdot D_{\text{step}} \\ D_{\text{step}} & = D_{\text{xbar}} + D_{\text{ctrl}} \\ D_{\text{xbar}} & = T_{\text{sw}} + D_{\text{nw}} \end{cases} \tag{12}$$

where D_{ble} is the product of execution step number N_{step} and the delay of a single step D_{step}. D_{step} is the sum of crossbar delay D_{xbar} and that of the CMOS controller D_{ctrl}. The nanowire delay D_{nw} is modelled by Eq.10 and D_{ctrl} is provided by the synthesis tool (e.g., Cadence RTL Compiler).

V. EVALUATION

To evaluate the performance of proposed architectures, a benchmark suite was synthesized and the their area and delay characteristics were compared with state-of-the-art FPGA. First, will present experimentation setup, followed by results and discussion, and finally limitations of this work.

TABLE II: FPGAs for Comparison

FPGA	CLB			Routing Architecture	
	BLE	SM	Config. Memory	Programmable Interconnect	Config. Memory
Baseline	CMOS	CMOS	6T-SRAM	CMOS	SRAM
MemFPGA	CMOS	CMOS	1T1R-RRAM		
MFPGA	MRL	MRL	3T2R-RRAM		
RFPGA	RBL	MRL	3T2R-RRAM		

TABLE III: Parameters of FPGA Architecture and Technology

Parameter	Description	Value
	Island-Style FPGA Architecture [10,16]	
K	No. of LUT inputs, $3 \leq K \leq 12$	–
N	No. of LUTs in a CLB, N=1,4,8	–
I	No. of CLB inputs, $I = \frac{K}{2}(N+1)$	–
$F_{\text{c,in}}$	Input connectivity fraction of each CLB	0.5
$F_{\text{c,out}}$	Output connectivity fraction of each CLB	$\frac{1}{N}$
L	Channel segment length (i.e., number of CLBs)	4
	Technology	
	Memristor (TaO_x RRAM) [18,22]	
F (nm)	Feature size	90
T_{sw} (ns)	Switching time (max of SET and RESET)	0.2
R_{L} (kΩ)	ON Resistance	200
R_{H} (MΩ)	OFF Resistance	200
R_{s} (MΩ)	Resistance of R_{s} (for RBL)	1
	Memory Cell Area [4]	
A_{sram} (F^2)	Area of a 6T-SRAM Cell	140
A_{1t1r} (F^2)	Area of a 1T1R-RRAM Cell	6
A_{3t2r} (F^2)	Area of a 3T2R-RRAM Cell	18
A_{m} (F^2)	Area of a memristor in crossbar	4
	Nanowire (Copper) [23]	
C_{nw} (fF/μm)	Capcitance in unit length	0.26
R_{nw} (Ω/μm)	Resistance in unit length	9.88
	CMOS: Synopsys EDK 90nm Lib	

A. Experiment Setting Up

Table II summarizes the characteristics of all FPGAs used in this sections. FPGA that employs SRAM as configuration memories is used as *baseline* implementation. In addition to traditional SRAM based FPGA, an MemFPGA that replaces SRAM of CLB with 1T1R as described in [20] was implemented and compared to the proposed architectures. All FPGAs in Table II can use routing architecture based on either CMOS [10] or RRAM [7,8]. To highlight the improvement of CLBs, all FPGAs use the same CMOS programmable interconnect.

This experiment uses classical island-style FPGA architecture [10] and Toronto 20 benchmark package [21] which consists of 20 benchmark circuits frequently used in different domains. Different numbers of LUT inputs (K=3–8,10,12) and numbers of LUTs within a CLB (N=1,8) are evaluated using area and delay model described in Section IV. All area and delay various reported are the average for all benchmark circuits synthesized. Since circuit 'tseng' of Toronto 20 cannot be synthesized correctly by ABC, it was not included in this experiment. Table III summarizes parameters of FPGA architecture and technology used in the experiments.

B. Results

Area Fig. 11 shows the area required for all four FPGAs. MFPGA and RFPGA typically need smaller area to implement CLBs (see Fig. 11(a)) whereas all need almost similar area to implement their routing architectures. Overall, MFPGA and

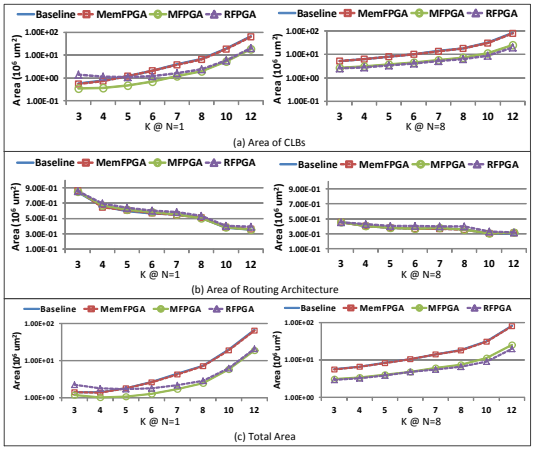


Fig. 11: Area of Different FPGAs

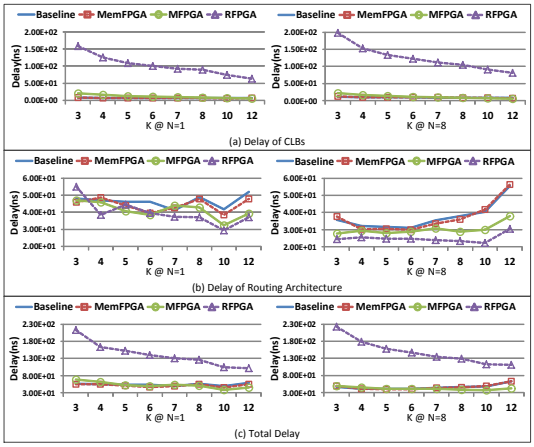


Fig. 12: Delay of Different FPGAs

RFPGA outperform the baseline and MemFPGA as logic area dominates the total area. Therefore, MFPGA and RFPGA provide a great potential to improve logic integration density.

Delay Fig. 12 shows the delay of all four FPGAs. The delay of the CLBs within MFPGA is similar to baseline and MemFPGA in case $K \geq 6$. On the other hand, the delay of CLBs within RFPGA are longer than others as each CLB needs several steps to complete its function. The delay of routing architecture in MFPGA and RFPGA are less than the other two FPGAs. Overall, MFPGA performs better than others in case $K \geq 6$ in all three clustering configurations. It is worth noting that each CLB of RFPGA can store data at run time, and hence its LUTs can be pipelined to improve its throughput.

In addition, MFPGA and RFPGA can be further improved if they incorporated with memristor-based routing architectures (e.g., [8]).

C. Limitations

This paper did not estimate power consumption of the proposed FPGAs as power modelling of memristor logics are still not mature [4]. Nevertheless, the proposed FPGAs may consume less power as memristors are non-volatile and hence they may consume less leakage power [3,4]. In addition, as this paper mainly illustrates the potential of FPGAs using memristor logic, technology challenges such as limited endurance, process variations [4] are out of the scope of this paper. These limitations will be studied in our future work.

VI. CONCLUSION

This paper proposed two novel FPGA implementations based on memristor logics. Their performances are intensively evaluated. Compared to the state-of-the-art, the proposed FPGAs provide a potential to improve the logic integration density, and possibly to reduce the delay. Hence, they are promising candidates for the future FPGA design and applications.

REFERENCES

[1] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.
[2] S. Hamdioui *et al.*, "Reliability challenges of real-time systems in forthcoming technology nodes," in *DATE*. IEEE, 2013.
[3] ITRS ERD report, 2010.
[4] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, 2013.
[5] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*. IEEE, 2015.
[6] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *NANOARCH*. IEEE, 2015.
[7] M. Liu *et al.*, "rfga: Cmos-nano hybrid fpga using rram components," in *NANOARCH*. IEEE, 2008.
[8] J. Cong et al., "Fpga-rpi: A novel fpga architecture with rram-based programmable interconnects," *TVLSI*, 2014.
[9] X. Tang *et al.*, "A high-performance low-power near-vt rram-based fpga," in *FPT*. IEEE, 2014.
[10] V. Betz et al., *Architecture and CAD for deep-submicron FPGAs*. Springer Science & Business Media, 2012.
[11] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, 2012.
[12] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, 2010.
[13] S. Kvatinsky et al., "Mrlmemristor ratioed logic," in *CNNA*. IEEE, 2012.
[14] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *ICCD*. IEEE, 2015.
[15] "Abc: A system for sequential synthesis and verification." [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/
[16] "Vpr and t-vpack: Versatile packing, placement and routing for fpgas." [Online]. Available: http://www.eecg.toronto.edu/ vaughn/vpr/vpr.html
[17] W. Zhao et al., "Design and analysis of crossbar architecture based on crs non-volatile memory cells," *JPDC*, 2014.
[18] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *AM*, 2011.
[19] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *JAP*, 1948.
[20] Y. Y. Liauw et al., "Nonvolatile 3d-fpga with monolithically stacked rram-based configuration memory," in *ISSCC*. IEEE, 2012.
[21] "Fpga place-and-route challenge." [Online]. Available: http://www.eecg.toronto.edu/ vaughn/challenge/challenge.html
[22] A. C. Torrezan et al., "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, 2011.
[23] D. B. Strukov *et al.*, "Cmol fpga: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, 2005.

6

NON-VON NEUMANN ARCHITECTURE

This chapter presents a non-Von Neumann computer architecture, Computation-In-Memory (CIM Architecture), for specific data-intensive applications (e.g., parallel addition and DNA sequencing). CIM is based on the integration of storage and computation in the same physical location using memristor technology. It significantly reduces the data communication between processors and memory by computing data directly within memory.

The content of this chapter consists of the following research articles:

1. *S. Hamdioui,* ***L. Xie****, H.A. Du Nguyen, M. Taouil, K.L.M. Bertels, Memristor Based Computation-in-Memory Architecture for Data-Intensive Applications, Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, March, 2015, pp. 1718-1725*

6.1. INTRODUCTION

One of the most critical challenges for the emerging data-intensive applications is the data storage and processing. The increase of the data size has already surpassed the capabilities of today's computation architectures suffering from the limited memory bandwidth, programmability overhead, energy inefficiency, and limited scalability. Therefore, it is necessary to develop a new resistive computing architecture for data-intensive applications.

6.2. MAIN CONTRIBUTIONS

The main contributions in the above aspects are as follows.

Development of CIM architecture [11]: CIM architecture is based on the integration of storage and computation in the same physical location using the non-volatile memristive devices. Such unique features lead to the reduction of memory accesses and leakage power consumption and the massive processing parallelism. Therefore, it shows a significant potential in addressing data-intensive problems than today's computer architectures (e.g., multi-core CPUs) in terms of computation efficiency. To illustrate how CIM architecture significantly advances the state-of-the-art, the performance of CIM and the conventional architectures for healthcare (i.e., DNA sequencing) and mathematics (i.e., parallel addition) applications are estimated. Both applications clearly show that the improvements are orders of magnitude. My major work is designing memristive device based logic circuits and estimating their performance in terms of area, delay and energy.

Memristor Based Computation-in-Memory Architecture for Data-Intensive Applications

[1]Said Hamdioui [1]Lei Xie [1]Hoang Anh Du Nguyen [1]Mottaqiallah Taouil [1]Koen Bertels [2]Henk Corporaal [2]Hailong Jiao

[1]Computer Engineering
Delft University of Technology
Delft, the Netherlands
First_Letter_First_Name.LastName@tudelft.nl

[2]Electronic Systems group
Eindhoven University of Technology
Eindhoven, The Netherlands
{h.corporaal, H.Jiao}@tue.nl

[3]Francky Catthoor [3]Dirk Wouters

[3]IMEC, Kapeldreef 75, B-3001
Leuven, Belgium
<catthoor@imec.be>

[4]Linn Eike

[4]RWTH Aachen University
Aachen, Germany
linn@IWE.RWTH-Aachen.de

[5]Jan van Lunteren

[5]IBM Research Laboratory
Zurich, Switzerland
jvl@zurich.ibm.com

***Abstract*—One of the most critical challenges for today's and future data-intensive and big-data problems is *data storage and analysis.* This paper first highlights some challenges of the new born Big Data paradigm and shows that the increase of the data size has already surpassed the capabilities of today's computation architectures suffering from the limited bandwidth, programmability overhead, energy inefficiency, and limited scalability. Thereafter, the paper introduces a new *memristor-based architecture* for data-intensive applications. The potential of such an architecture in solving data-intensive problems is illustrated by showing its capability to increase the computation efficiency, solving the communication bottleneck, reducing the leakage currents, etc. Finally, the paper discusses why memristor technology is very suitable for the realization of such an architecture; using memristors to implement dual functions (storage and logic) is illustrated.**

I. INTRODUCTION

Today's applications are becoming extremely data intensive; healthcare, social media, large scientific/engineering experiments, and security are just couple of examples. As the speed of information growth exceeds Moore's Law, since the beginning of this new century, excessive data is posing major challenges [1] and a new scientific paradigm is born: data-intensive scientific discovery, also known as *Big Data problems*. The primary goal is to analyse and increase the understanding of both data and processes in order to extract the highly useful information hidden in the huge volume of data, which in turn can be used to increase e.g., the productivity. Storing and analysing such data is posing major challenges as the data volume already surpassed the capability of today's computers which suffer from e.g., communication and memory-access bottlenecks due to limited bandwidth [2-lahiri, 3-somavat]. For instance, a transfer of 1 petabyes data at a rate of 1000MB/second will take 12.5 days! Memory size and memory access do not only kill the performance, but also severely impact energy/power consumption [2, 3 ,4]. In addition, CMOS technology used to implement today's architectures contributes to such consumption due to high leakage currents; not to mention other challenges the technology is facing such as limited scalability, reduced reliability [30-34], etc. In conclusion, today's CMOS based architectures are not able to provide the computation capability needed for data-intensive applications. *New* architectures based on *new* technologies are urgently required.

This paper discusses a new architecture, *Computation-In-Memory* (CIM Architecture), for specific data-intensive applications; it is based on the *integration* of *storage* and *computation* in the *same* physical location (crossbar topology) and the use of *non-volatile resistive-switching technology (memristive devices or memristors in short)* [30, 38, 39, 94] instead of CMOS technology.

The rest of the paper is organized as follows. Section II highlights the Big Data problem and shows how the conventional computers based on CMOS technology are incapable to deal with such problems; and motivates the need for a new architecture. Section III discusses CIM architecture, including its concept and its potential; the section puts that in perspective by taking couple of application examples and comparing the performance of CIM architecture with the state-of-the-art. Section III shows why memristor is the key enabler for CIM architecture by illustrating how the device, in crossbar architecture, can perform a dual function (storage and computation). Section IV concludes the paper.

II. DATA-INTENSIVE APPLICATIONS VS CMOS COMPUTERS

A. Big Data and Data Intensive Applications

No one can deny the fact that a large number of fields and sectors, ranging from economics and business activities to public administration, from national security to many scientific research areas, involve data-intensive applications, hence, dealing with Big Data problems. Big Data is extremely valuable to generate productivity in businesses and evolutionary breakthroughs in scientific disciplines, which give us a lot of opportunities to make great progress in many fields [1]. The primary goal is to increase the understanding of processes in order to extract so much potential and highly useful values hidden in the huge volumes of data, and therefore, it comes with many challenges, such as data capture, *data storage, data analysis,* and data visualization.

978-3-9815370-4-8/DATE15/©2015 EDAA

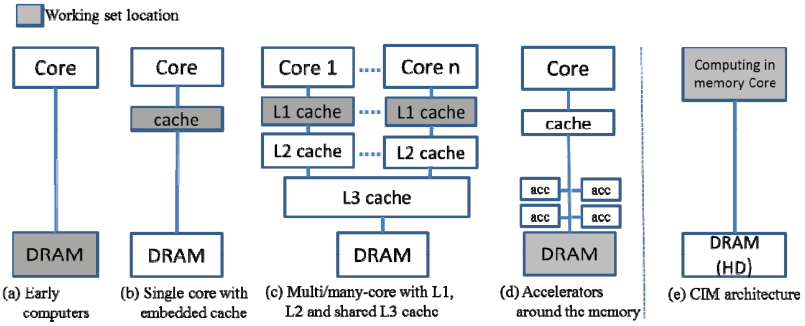


Figure 1: Classification of computing systems based on working set location

Performing data analysis within economically affordable time and energy is the *pillar* to solve big data problems

B. Today's Computers

The computing systems, developed since the introduction of stored program computers by John von Neumann in the forties [5], can be classified based on the location of the so-called "*working set*" (loosely defined as the collection of information referenced by a program during its execution) into four classes (a) to (d) as shown in Figure 1. In the early computers (typically before the 80s), the working set was contained in main memory. Due to the gap between the core (CPU) speed and the memory, caches were introduced to reduce the gap and increase the overall performance, where the caches have become the location of the working set. Today's computing systems for data-intensive applications are still based on Von Neumann (VN) architectures and still rely on many parallel (mini-)cores with a shared SRAM cache (parallel CPUs, GPUs, SIMD-VLIWs, vector processors); see Figure 1(c). Clusters of cores can be replicated many times, each having their own L1 cache, but it is far from realistic to assume a distributed reasonable sized L1 cache in every mini-core; too much area and leakage power overhead is incurred in that case. Such solutions suffer from major limitations such as a decreased performance acceleration per core [6], increased power consumption [7, 8], and limited system scalability [6, 9]. These are mainly caused by the processor-memory bottleneck [10, 11]. As current data-intensive applications require huge data transfers back and forth between processors and memories through load/store instructions [12], the maximal performance cannot be extracted as the processors will have many idle moments while waiting for data [10-14]. Computation, which is the main activity of a system, by far consumes less energy and chip area, and has lower execution time compared to communication and memory access (e.g., L1 cache), especially for data intensive applications [15]. The energy consumption of the cache accesses and communication makes up easily 70% to 90% [2,3,4]; not to mention the rest of the memory hierarchy. In addition, programmability in conventional processors also comes at a substantial energy cost: for example, [4] reports that executing a *multiply* instruction on a simple in-order core in 45nm technology consumes about 70 pJ, whereas the actual operation itself consumes less than 4 pJ. The overhead is due to instruction fetching and decoding and other control.

Triggered by these issues, the design of high-performance computing systems is starting to move away from a conventional computation-centric model towards a more data-centric approach. The latter concept intends to improve performance and power efficiency through reduction of data movement by performing the actual processing closer to where the data resides in the memory system. Several alternative architectures are proposed that fall into this category. One alternative is called "*Processor-in-memory*" as shown in Figure 1(d); additional processing units (accelerators) are put around one or more memories which are the working set location; examples are FlexRAM [16], DIVA [17], TeraSys [18], EXECUBE [19], HTMT [20], Computational RAM [21], DSP-RAM [22], Smart memories-based architecture [23], Gilgamesh [24], Continuum computer architecture [24], and MICRON's architecture for automata processing [26]. The second alternative architecture is called **"***Memory-in-processor*", which is an extension of what is shown in Figure 1(c), where extra addressable memories are put close to the cores; examples are Data Arithmetic SRAM [27] and Connection machine [28]. The third is called "*In memory computing/database*" (mainly for database management), which primarily relies on the storage of the complete database working set in the main memory of dedicated servers rather than relying on complicated relational databases operating on comparatively slow disk drives [29].

C. CMOS Technology

Today's computers are manufactured using the traditional CMOS technology, which is reaching the inherent physical limits due to down-scaling. Technology nodes far below 20nm are presumably only practical for limited applications due to multiple challenges [30-34,], such as high static power consumption, reduced performance gain, reduced reliability, complex manufacturing process leading to low yield and complex testing process, and extremely costly masks.

Many novel nano-devices and materials are under investigation to replace the CMOS technology in next IC generations. Among the emerging devices, such as graphene transistor [35], nanotube [36], tunnel field-effect transistor (TFET) [37], etc., memristor [38, 39] is a promising candidate.

Its advantages are CMOS process compatibility [40], lower cost, zero standby power [41], nanosecond switching speed [42], great scalability and high density [43], and non-volatile nature [44, 45]. It offers a high OFF/ON resistance ratio [46] and it is promising to have a good endurance and retention time [47]. More importantly, the memristor is a *two-terminal resistive-switching device* that can be used to *build both storage and information processing units* [48, 49, 50].

D. The Need of New Architecture

The speed at which data is growing has already surpassed the capabilities of today's computation architectures suffering from communication bottleneck (due to limited bandwidth), energy inefficiency (due to CMOS technology), and programmability overhead. Therefore, there is a need for a new architecture using new device technology, being able to (a) eliminate the communication bottleneck and support massive parallelism to increase the overall performance, (b) reduce the energy inefficiency to improve the computation efficiency. This can be done by taking the data-centric computing concept much further by integrating the processing units and the memory in the same physical location and therefore moving the working set into the core as shown in Figure 1 (e).

III. CIM ARCHITECTURE- BEYOND VON NEUMANN

This section presents first the concept of the CIM architecture as an alternative of today's architectures. Thereafter the potential of the architecture will be illustrated by selecting couple of data-intensive applications and making an analysis of different performance metrics and comparing the results with the state-of-the art. Finally, major open questions related to the implementation of the CIM architectures will be highlighted.

A. CIM Architecture Concept

To tackle the big data computation problems and solve the todays computers bottlenecks, we propose a memristor-based architecture paradigm where both the computation and the storage take place at the same physical location (the crossbar array). The approach intends to provide solutions based computing-in-memory architectures using non-volatile devices. Figure 2 shows the traditional versus the proposed CIM architecture; note that in CIM architecture the storage and computation are integrated together in a very dense crossbar array where memristors are injected at each junction of the crossbar (top electrode and bottom electrode). The communication and control from/to the crossbar can be realized using CMOS technology. CIM architecture addresses important challenges and has huge potentials which go substantially beyond the current state-of-the-art.

- *Tightly integrated computation-in-memory crossbar architecture supporting massive parallelism*: as the storage and computation are integrated together, the communication bottleneck is significantly reduced. In addition, because the memristor technology is highly scalable (~5nm [30]), huge crossbar architectures allowing massive parallelism are feasible.
- *An architecture with practically zero leakage:* Today's architectures heavily rely on SRAMs as caches. These are required to have a very fast R/W access, leading to increasingly high leakage with technology scaling. Hence, the memristor crossbar architecture solves also the leakage bottleneck, at least in the memory.
- *Significant performance improvement at lower energy and area:* Given the nature of the architecture (supporting massive parallelism), the non-volatile technology in the crossbar, and the small feature size of the memristor, the architecture has the potential of improving the overall performance at extremely low power consumption and smaller area. The next section illustrates this potential for two different applications.

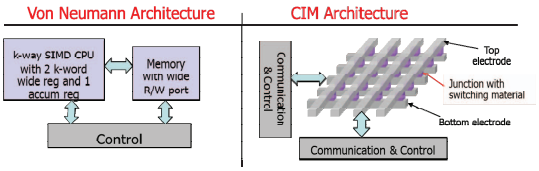


Figure 2: Traditional versus proposed architecture

B. CIM Architecture Potential

To illustrate how CIM architecture significantly advances the state-of-the-art, the performance of CIM and the conventional architectures for two applications will be estimated.

1) Healthcare: using genomics in diagnosing/treating diseases: the continuously dropping price of DNA sequencing has shifted the challenge from acquiring genetic information to the actual processing and analysis of this information [51]. Despite its computational simplicity, the huge amount of genetic data (hundreds of GBs per experiment) that needs to be processed makes the analysis rather time consuming and even not practical due to communication/memory access bottleneck. A practical solution used today for comparing two DNA sequences is based on the creation of a sorted index of the reference DNA that can be used to identify the location of matches and mismatches in another sequence rapidly. This approach, however, results in eliminating available data locality in the reference and causing huge number of cache misses with high memory access penalty and high energy cost. To quantify this effect, we assume we have 200 GB of DNA data to be compared to a healthy reference of 3GB for 50% coverages [51] and evaluate the DNA sorted-index sequencing algorithm on both the conventional architecture and CIM architecture; see Figure 2. The conventional architecture is assumed to scalable multi-core architecture, consisting of a number of clusters, each with 32 cores. Table 1 presents further made assumptions for both architectures. Three metrics are used for the evaluation: (a) the energy-delay product per operations, (b) the computation efficiency defined as the number (#) of operations per required energy, and (c) performance (#operations) per area;

Table 1: Assumptions made for conventional and CIM architectures

Assumption for conventional architecture	Assumptions for CIM architecture

Generic assumptions
• FinFET 22nm multi-core implementation
o Gate delay = 14 ps [53, 54]
o Area per gate: 0.248 μm^2 [30]
o Power consumption per gate: 175 nW [54]
o Leakage power: (a) Leakage power consumption per gate: 42,83 nW [30], (b) Leakage duration: cycle time – delay per gate
o Operating frequency: 1 GHz
• The architecture consist of a certain number of clusters of processing units, each cluster shares an 8kB L1 cache.
For Healthcare example
• Typically, the DNA reference sequence must be covered 50 times by short reads. The length of the short reads are assumed to be 100 characters.
• 200GB of DNA data is compared to a healthy reference of 3GB.
• Number of short reads no_short_reads = coverage * 3 * giga/ short_read_len; coverage=50, short_read_len=100
• Number of comparisons no_comparsions = 4*no_short_reads, for each A, C, G, T nucleotides
• Number of clusters is 18750, each contains 32 comparators.
o Limited with the state-of-the-art chip area
• Each cluster shares 8 kB Cache (per cluster)
o Area: 0.0092 mm^2 [57]
o Hit ratio = 50%; Hit cycle time = 1 cycle
o Miss penalty = 165 cycle [55];
o Write cycle time = 1 cycle;
o Static power: 1/64 Watt [56]
For Mathematics example
• Fully scalable *reusing* clusters; each has 8 kB shared cache.
• Additions are performed by 32 adders per cluster:
o Adder architecture: Carry Look Ahead (CLA)
o Number of gates per adder: 208 [52]
o Number of gate delay: 18;
o Adder latency: 252ps = 18*14ps
o Energy per 32-bit adder
• Shared 8 kB Cache (per cluster): the same as for healthcare except with 98% hit rate,
	Generic assumptions
	• Memristor 5nm crossbar implementation [30]
	o Memristor write time: 200 ps [60]
	o Area per memristor: $1x10^{-4}$ um^2 [30]
	o Dynamic energy per write operation: 1 fJ [30]
	• The memory capacity of the CIM architectures is assumed to be equal to the sum of all caches for the CMOS based computer.
	For Healthcare example
	• Each comparison is performed by a comparator
	o Comparator: 2 XOR and a NAND implemented by implication logic [58]
	o Number of memristors per comparator: 13 (XOR: 5, NAND: 3)
	o Area per comparator: $1.3 * 10^{-3}$ um^2 [58]
	o Number of steps per comparator: 16 steps (Two XOR works in parallel, an XOR takes 13 steps, and an NAND takes 3 steps, step takes a memristor write time). [58]
	o Comparator latency: 3.2 ns
	o Dynamic energy per comparator: 45fJ [58]
	o Static energy per comparator: 0 fJ [30]
	• The crossbar size equals to total cache size of CMOS computer
	o Size= 18750*8kB = 1.536*10^8 memristors
	o Date hit rate = 50%, Hit cycle time = 1 cycle
	o Miss penalty = 165 cycle
	For Mathematics example
	• The crossbar is scalable to support the 10^6 adders
	• Additions are performed by memristors
	o Adder architecture: TC-adder [59]
	o Number of memristors per adder: 34 (N+2, N=32) [59]
	o Area per adder: $3.4x10^{-3}$ um^2
	o Number of steps per 32-bit addition: 133 (4N+5, N=32, each step takes a memristor write time). [59]
	o Adder latency: 16600 ps (133 * 200 ps)
	o Dynamic energy per 32-bit adder: 246 fJ (8 (operations per bit) *32 (bits) * 1 fJ [59])
	o Static energy per 32-bit adder: 0 fJ [30]
	• The memory hit rate is assumed to be 98%, remaining parameters are the same as for the healthcare example.

2) Mathematics: Here we assume 10^6 parallel addition operations and make similar assumptions for the two architectures as those done in the previous example; see Table 1 for the details.

Table 2 shows the obtained results for the two applications for conventional (Conv.) and CIM architectures; both applications clearly show that the improvements are orders of magnitude. Reducing/eliminating memory accesses, the non-volatile technology, and the high parallelism are enabling these improvements.

Table 2: Huge potential of CIM architecture

Metric	Archit.	DNA Sequencing	10^6 additions
Energy-delay/ operations	Conv.	2.0210e-06	1.5043e-18
	CIM	2.3382e-09	9.2570e-21
Computing efficiency	Conv.	4.1097e+04	6.5226e+09
	CIM	3.7037e+07	3.9063e+12
Performance area	Conv.	5.7312e+09	5.1118e+09
	CIM	5.1118e+09	4.9164e+12

C. CIM Architecture Challenges

Although we mentioned that CIM architecture targets data-intensive applications, especially applications that require massive parallelism and huge data working sets to be continuously kept in the memory, the proposed concepts can be adapted to any computation-in-memory (CIM) architecture for high computation efficiency. This architecture paradigm shift, based on memristor technology, changes the traditional system design, compiler tools, manufacturing processes, etc., to facilitate its "industrialization". In fact, it is well recognized that memristor technologies are very promising. Although understanding of its capabilities and limitations is still evolving, the technology is expected to rule the computer world, from material science (understanding and proving the properties of the materials and assuring reliability), to design methods, tools, operating systems and its potential applications. Examples of its use are replacement of RAM, flash and even disk drives, complex self-learning neural networks, advanced artificial neural brains, and many more [61]. It may play a significant role in advancing Exascale

computing, 'computer on a chip' capabilities, as well as driving developments in neural and analogue computing. Next section will elaborate more on memristor technology.

IV. MEMRISTOR - THE KEY ENABLER FOR CIM ARCHITECTURE IMPLEMENTATION

This section reviews first the memristor technology. Thereafter its suitability for the realization of both storage and logic functions is discussed and illustrated.

A. Memristor Technology

Memristors or memristive devices, also referred to as resistive memory devices, are very broad groups of memory technologies; they can be classified based on their dominant physical operating mechanism into three classes [30]: *Phase Change Memories, Electrostatic/ Electronic Effects Memories, and Redox memories.* The redox-based resistive switching devices (ReRAMs) are attracting most attention due to their excellent scaling, endurance, and retention properties [30, 95]; their physical mechanism for switching is based on reduction/oxidation (Redox)-related chemical effects. The category of "Redox RAM" encompasses a wide variety of *Metal-Insulator-Metal (MIM)* structures; the electrochemical mechanisms driving the resistance state (from high to low or vice versa) can operate in the bulk I-layer, along conducting filaments in the I-layer, and/or at the I- layer/metal contact interfaces in the MIM structure. The ReRAMs consist of three types, two bipolar and one unipolar [30,50,61]; the rest of the section will focus on the two bipolar devices and show the best available properties for both device types; these are the Valence Change Memory (VCM) and the Electrochemical metallization (ECM) devices.

For both VCM (HfO_x) and ECM (Ag-chalcogenide) devices a feature size of $F = 10$ nm was reported [62, 63]. A minimum switching time of < 200 ps was shown for TaO_x-based VCM devices [42], whereas for ECM devices (Ag-MSQ) switching times below 10 ns were realized [64]. In terms of endurance, more than 10^{12} cycles are feasible for TaO_x-based VCM cells and more than 10^{10} for Ag-GeSe ECM cells [65]. Extrapolated retention of > 10 years was, for example, shown in [66] (TaO_x-based VCM cells) and [67] (Ag-chalcogenide).

In ECM devices a conductive metallic filament (Cu or Ag) is established during switching, thus, the filament length can be considered the state variable [68]. For a memristive ECM model, both electronic and ionic currents must be considered, and the strong non-linearity of the switching kinetics must be reflected by the model. VCM modelling is even more challenging due to the versatile device physics [69]. Since simple memristor models fail to predict the correct device behaviour [39, 70], more complex empirical and physics-based models were developed recently [71, 72].

B. Memristor for Crossbar Memories

The primary driver for ReRAM research is the semiconductor industry seeking for novel energy-efficient non-volatile and highly scalable memory elements [30]. A straightforward implementation of the ReRAM array is realised using a passive crossbar architecture, resulting in the highest density [73, 74]. However, this architecture suffers from undesired paths for current called *sneak paths* [75]; due to the low resistive current paths, the maximum array is limited to small arrays [76]. To overcome this issue, three classes of solutions are proposed:

- *Selector devices*, which are separate devices in connection with the RRAM cell such as a diode or a transistor (1S1R) [77, 78].
- *Switching device modification*, where the resistive devices is modified; E.g., serially connecting of two anti-serial memristive devices (bipolar switches) resulting into a "complementary resistive switcher" (CRS) being able to block the current at low voltage irrespective of the state of the device [78], or the deployment of a high nonlinear memristive device (due to current-controlled negative differential resistance) to overcome sneak path [79].
- *Bias schemes*, where the voltage bias applied to non-accessed wordlines and bitlines are set to values different from those applied to accessed wordline and bitlines in order to minimize the sneak path current; examples are multistage reading [80] and use of AC signal instead of DC for sensing the data stored in the desired cell [81].

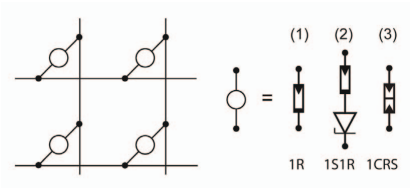


Figure 3: Illustration of complementary resistive switch

Figure 3 sketches the concept of the the crossbar array and some junction options to deal with sneak paths, while Figure 4 illustrates the *I-V* characteristic of a CRS cell which consists of two memristive ECM devices A and B. The states '0' and '1' are the logical storage states and the state 'LRS/LRS' occurs only when reading the memory state. The internal memory states '0 'and '1' of a CRS cell are indistinguishable at low voltages because state '0' as well as state '1' show a high resistance. Therefore, no parasitic current sneak paths can arise. To read the stored information of a single CRS cell, a read voltage must be applied to the cell. If the CRS cell is in state '0', then it switches to state 'ON'; if the cell is in state '1' then it remains in its state. In case conventional crossbar (with resistive current paths), reading ON state is a destructive operation, therefore, it is necessary to write back the previous state of the cell after reading it. In general, the writing of state '0' requires a negative voltage ($V < V_{th,4}$) and for writing '1' a positive voltage $V > V_{th,2}$ is required.

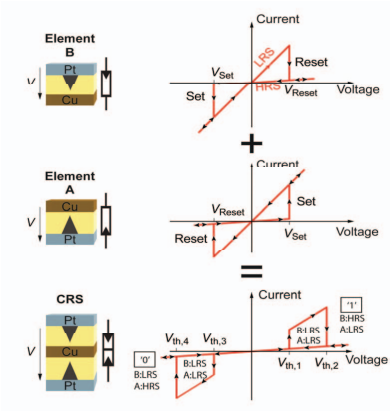


Figure 4: Left: Zoom in a passive nano-crossbar array. Right: possible cross point junctions

C. *Memristor for Logic Functions*

Memristive devices are well suited for the implementation of logic functions including: (a) programmable interconnects [82], (b) look-up tables (LUTs) [83] or content addressable memories (CAMs) [84], and (c) sequential 'stateful' logic operations [49, 58, 85].

Programmable logic arrays based on resistive switching junctions were suggested first in [82] and later also applied to FPGAs [86]. Typically, the CMOS overhead is relatively large since the array size is small. A next step was the CMOL FPGA concept [87], where a sea of elementary CMOS cells is connected to a small crossbar part-array. In this approach the elementary CMOS cells are connected via resistive switches (1S1R) enabling wired-or functionality. In general, reconfigurable on-chip wiring enables new options for memristive chip design and can also be combined with the functionalities as those described next .

Resistive memories can be either used to implement small LUTs for FPGAs (as suggested in [83]) or LUTs can be mapped to large-scale crossbar arrays [88, 89] to reduce the crossbar array overhead. Moreover, CAMs based on memristors are feasible with different flavors [90, 91]; e.g., a CRS-based CAM is recently demonstrated [84].

Memristors are also used to design (sequential) logic operations based on Boolean functions [92] or (material) implication logic (IMP) [49, 58, 85]; the latter seems to be more popular. Figure 5 uses two examples to illustrate the concept of IMP. Figure 5(a) gives a basic logic function using two memristors. Together with a load resistor R_G, the operation p IMP q is conducted as follows [49]:

1. Set device p to p ($V_P = \pm V_{Write}$)
2. Set device Q to q ($V_Q = \pm V_{Write}$)
3. $q' = p$ IMP q ($V_P = V_{COND}$ and $V_Q = V_{write}$)
4. Read q'

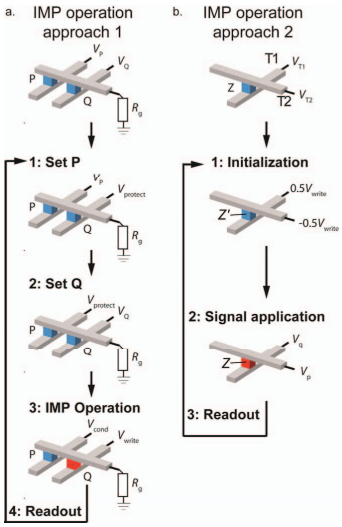


Figure 5: Two ways to implement IMP. Blue cube represents state '0' and the red cube state '1'

An alternative approach to implement p IMP q, with superior performance, is suggested in [93], as shown in Figure 5(b). The input signals Vp = $\pm$½V_{Write} and Vq = $\pm$½V_{Write} are applied at the terminals T1 and T2 of the memristor. The final result is stored as resistive state Z. For Z = p IMP q the following steps are performed:

1. Init device Z to '1' (VT1 = +½V_{Write}, VT2 = -½V_{Write})
2. Z' = p IMP q (VT1 = Vq, VT2 = Vp)
3. Read Z'

IMP can be used to design arithmetic operations such as adders [58, 56]; hence, it paves the path to more complex memristive in-memory-computing architectures.

V. CONCLUSION

This paper discusses data storage and analysis as one of the most critical challenges for today's and future data-intensive and big-data problems. It shows how the increase of the data size has already surpassed the capabilities of today's computation architectures suffering from the limited bandwidth, energy inefficiency and limited scalability. Thereafter, the paper proposes a new architecture based on the integration of the storage and computation in the same physical location (using a crossbar topology); the architecture is driven by non-volatile resistive-switching technology (memristors) instead of traditional CMOS technology. Therefore, it has the potential to reduce both the memory wall and energy consumption with orders of magnitude, and enables massive parallelism. Hence, significantly improving the performance and enabling the solution of big-data problems. The details and many aspects of the architecture still need to be worked out.

ACKNOWLEDGMENT

We would like to thank all people who contributed to the discussion of CIM architectures including Zaid Al-ars from delft University of Technology, Jan van Dalfson from Eindhoven University of Technology, Luca Benini from ETHZ, Shahar Kvatinsky from Technion, Lotfi Mhamdi from Leed University, Albert Cohen from INRIA, Stephan Menzel and Vikas Rana from RWTH, and Andrea Fantini from IMEC.

REFERENCES

[1] P. Chen and C-Y Zhang, 'Data-intensive applications, challenges, techniques and technologies: A survey on Big Data', Information Sciences, v 275, pp. 314-347, 2014

[2] K. Lahiri, A. Raghunathan, 'Power analysis of system-level on-chip communication architectures', International Conference on Hardware/Software Codesign and System Synthesis, CODES + ISSS, pp 236-241, 2014

[3] P. Somavat and V. Namboodiri. 'Energy consumption of personal computing including portable communication devices', Journal of Green Engineering, 1(4):447–475, 2011

[4] M. Horowitz, 'Computing's Energy Problem and what we can do about it', slides of the keynote at ISSCC 2014

[5] A.W. Burks, et al., 'Preliminary discussion of the logical design of an electronic computing instrument', 1946

[6] H. Esmaeilzadeh, et al., 'Dark silicon and the end of multicore scaling', in Proceedings of the 38th annual international symposium on Computer architecture, pp. 365-376, 2011

[7] S. Bampi, and R. Reis, 'Challenges and Emerging Technologies for System Integration beyond the End of the Roadmap of Nano-CMOS', 'VLSI-SoC: Technologies for Systems Integration' (Springer Berlin Heidelberg), pp. 21-33, 2011

[8] T. Yuan, 'CMOS design near the limit of scaling', IBM Journal of Research and Development, 2002, 46, (2.3), pp. 213-222

[9] X.-J. Yang et al., 'Progress and Challenges in High Performance Computer Technology', J. Comp. Sci. Tech., 2006, 21, (5), pp. 674-681

[10] S. A. McKee, 'Reflections on the Memory Wall', CF'04, pp. 162, 2004.

[11] M.V. Wilkes, 'The Memory Wall and the CMOS End-point', SIGARCH Comput. Archit. News, 1995, 23, (4), pp. 4-6

[12] R.E. Bryant, 'Data-Intensive Scalable Computing for Scientific Applications', Computing in Science & Engin., 2011, 13, (6), pp. 25-33

[13] W.A. Wulf and S.A. McKee, 'Hitting the memory wall: implications of the obvious', SIGARCH Comp. Archit. News, 23, pp. 20-24, 1995

[14] D. Patterson et al. 'A case for intelligent RAM', IEEE Micro, 1997, 17, (2), pp. 34-44

[15] C. Hernandez et al., 'Energy and Performance Efficient Thread Mapping in NoC-Based CMPs under Process Variations', in International Conference on Parallel Processing, 2011, pp. 41-50

[16] Y. Kang et al., 'FlexRAM: Toward an advanced Intelligent Memory system', in IEEE 30th International Conference on Computer Design, pp. 5-14, 2012

[17] J. Draper et al., 'The architecture of the DIVA processing-in-memory chip', In Proceedings of the 16th international conference on Supercomputing, pp. 14-25, 2002

[18] M. Gokhale et al., 'Processing in memory: the Terasys massively parallel PIM array', Computer, vol. 28, pp. 23-31, 1995

[19] P. M. Kogge, 'EXECUBE-A New Architecture for Scaleable MPPs', in International Conference on Parallel Processing, Vol. 1, 1994, pp. 77-84

[20] P. M. Kogge et al., 'PIM architectures to support petaflops level computation in the HTMT machine', in International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 1999, pp. 35-44

[21] D. G. Elliott et al., 'Computational RAM: implementing processors in memory', IEEE Design Test of Computers, vol. 16, pp. 32-41, 1999

[22] Z. Wang et al., 'DSP-RAM: A logic-enhanced memory architecture for communication signal processing', in IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1999, pp. 475-478

[23] M. Ken et al., 'Smart Memories: a modular reconfigurable architecture', in Proceedings of the 27th International Symposium on Computer Architecture, 2000, pp. 161-171

[24] T. L. Sterling and H. P. Zima, 'Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing', in ACM/IEEE Conference Supercomputing, 2002, pp. 48-48

[25] T. Sterling and M. Brodowicz, 'Continuum computer architecture for nano-scale and ultra-high clock rate technologies', in Innovative Architecture for Future Generation High-Performance Processors and Systems, 2005, pp. 1-9

[26] P. Dlugosch et al., 'An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing', IEEE Transactions on Parallel and Distributed Systems, 2014, vol. 99, pp. 3088- 3098

[27] N. Venkateswaran et al., 'Memory in Processor: A Novel Design Paradigm for Supercomputing Architectures', 2003, pp. 19-26

[28] L. W. Tucker and G. G. Robertson, 'Architecture and applications of the Connection Machine', Computer, vol. 21, pp. 26-38, 1988

[29] H. Plattner, 'SanssouciDB: An In-Memory Database for Processing Enterprise Workloads', Datenbanksysteme in Büro, Technik und Wissenschaft (German Database Conference), 2011

[30] ITRS ERD report. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm, 2010

[31] B. Hoefflinger, 'Chips 2020: A Guide to the Future of Nanoelectronics', The Frontiers Collection, Springer Berlin Heidelberg, 2012, pp. 421–427

[32] J. McPherson, 'Reliability trends with advanced CMOS scaling and the implications for design', in IEEE Custom Integrated Circuits Conference, 2007, pp. 405–412

[33] S. Borkar, 'Design perspectives on 22Nm CMOS and beyond', in Proceedings of the 46th Annual Design Automation Conference, 2009, pp. 93-94

[34] G. Gielen, et al., 'Emerging yield and reliability challenges in nanometer CMOS technologies', in Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1322-1327, 2008

[35] F. Schwierz, 'Graphene transistors', Nature nanotechnology, vol. 5, no. 7, pp. 487–496, 2010

[36] G. Agnus et al., 'Two-terminal carbon nanotube programmable devices for adaptive architectures', Advanced Materials, vol. 22, no. 6, pp. 702–706, 2010

[37] K. Boucart and A. M. Ionescu, 'Double-gate tunnel fet with high-gate dielectric', IEEE Transactions on Electron Devices, vol. 54, no. 7, pp. 1725–1733, 2007

[38] L. Chua, 'Memristor-the missing circuit element', IEEE Transactions on Circuit Theory, vol. 18, no. 5, pp. 507–519, 1971

[39] D. B. Strukov et al., 'The missing memristor found', Nature, vol. 453, no. 7191, pp. 80–83, May 2008

[40] D. B. Strukov et al., 'Hybrid cmos/memristor circuits', in IEEE International Symposium on Circuits and Systems, 2010, pp. 1967–1970

[41] H. Lee et al., 'Low-power and nanosecond switching in robust hafnium oxide resistive memory with a thin ti cap', IEEE Electron Device Letters, vol. 31, no. 1, pp. 44–46, 2010

[42] A. C. Torrezan et al., 'Sub-nanosecond switching of a tantalum oxide memristor', Nanotechnology, vol. 22, no. 48, pp. 1-7, 2011

[43] M.-J. Lee et al., 'A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5x/TaO2x bilayer structures', Nature Materials, vol. 10, no. 8, pp. 625–630, 2011

[44] T. Prodromakis and C. Toumazou, 'A review on memristive devices and applications', in 17th IEEE International Conference on Electronics, Circuits and Systems, 2010, pp. 934–937

[45] W. Zhao et al., 'Nanodevice-based novel computing paradigms and the neuromorphic approach', in IEEE International Symposium on Circuits and Systems, 2012, pp. 2509–2512

[46] Y. S. Chen et al., 'Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity', in IEEE International Electron Devices Meeting, 2009, pp. 1–4

[47] J. J. Yang et al., 'High switching endurance in tao memristive devices', Applied Physics Letters, vol. 97, p. 232102- 232103, 2010

[48] J. J. Yang et al., 'Memristive devices for computing', Nat. Nanotechnol., vol. 8, pp. 13-24, 2013

[49] J. Borghetti et al., ''Memristive' switches enable 'stateful' logic operations via material implication', Nature, 464, pp. 873-876, 2010

[50] S. Hamdioui et al., 'Memristor based memories: Technology, design and test', IEEE 9th International Conference on Design & Technology of Integrated Systems In Nanoscale Era, pp. 1-7, 2014

[51] E. A. Worthey, 'Analysis and annotation of whole-genome or whole-exome sequencing-derived variants for clinical diagnosis', Current Protocols in Human Genetics, 2001

[52] B. Parhami, Computer arithmetic: algorithms and hardware designs. Oxford University Press, Inc., 2009

[53] A. Muttreja et al., 'CMOS logic design with independent-gate FinFETs', 25th International Conference on Computer Design, pp. 560–567, 2007

[54] C. Meinhardt et al., 'FinFET basic cells evaluation for regular layouts', in IEEE Fourth Latin American Symposium on Circuits and Systems, pp. 1–4, 2013

[55] D. Levinthal, Cycle Accounting Analysis on Intel R CoreTM2 Processors. Intel Corp, 2008

[56] C.Y. Lee et al., 'CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations', in 48th Design Automation Conference, pp. 866–871, 2011

[57] E. Karl et al., 'A 4.6 GHz 162Mb SRAM design in 22nm tri-gate CMOS technology with integrated active V MIN-enhancing assist circuitry', in IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp. 230–232, 2012

[58] S. Kvatinsky et al., 'Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies', IEEE Transactions on VLSI Systems, vol. 22, no. 10, pp. 2054–2066, 2014

[59] A. Siemon et al., 'A Complementary Resistive Switch-based Crossbar Array Adder', arXiv:1410.2031, 2014

[60] J. Walker, Memristor and the future, [Online]. Available: http://www.nobeliefs.com/memristor.htm, 2010

[61] R. Waser et al., 'Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges', Advanced Materials, vol. 21, pp. 2632-2663, 2009

[62] B. Govoreanu et al., '10x10nm2 Hf/HfOx Crossbar Resistive RAM with Excellent Performance, Reliability and Low-Energy Operation', IEEE International Electron Devices Meeting, pp. 31.6.1-31.6.4, 2011

[63] K. Terabe et al., 'Quantized conductance atomic switch', Nature, vol. 433, pp. 47-50, 2005

[64] M. Meier et al., 'A Nonvolatile Memory With Resistively Switching Methyl-Silsesquioxane', IEEE Electron Device Letters, vol. 30, pp. 8-10, 2009

[65] M. N. Kozicki et al., 'Nanoscale memory elements based on solid-state electrolytes', IEEE Transactions on Nanotechnology, vol. 4, pp. 331-338, 2005

[66] Z. Wei et al., 'Retention model for high-density ReRAM', 4th IEEE International Memory Workshop, pp. 1-4, 2012

[67] M. Kund et al., 'Conductive bridging RAM (CBRAM): an emerging non-volatile memory technology scalable to sub 20nm,' IEEE International Electron Devices meeting Technical Digest, pp. 754 – 757, 2005

[68] S. Ferch et al., 'Simulation and Comparison of two Sequential Logic-in-Memory Approaches Using a Dynamic Electrochemical Metallization Cell Model', Microelectronics Journal, vol. 45, pp. 1416-1428, 2014

[69] R. S. Williams et al., 'Physics-based memristor models', IEEE International Symposium on Circuits and Systems, pp. 217-220, 2013

[70] E. Linn et al., 'Applicability of Well-Established Memristive Models for Simulations of Resistive Switching Devices', IEEE Transactions on Circuits and Systems, vol. 61, pp. 2402 - 2410, 2014

[71] M. D. Pickett et al., 'Switching dynamics in titanium dioxide memristive devices', Journal of Applied Physics, 2009

[72] J. P. Strachan et al., 'State Dynamics and Modeling of Tantalum Oxide Memristors', IEEE Transactions on Electron Devices, vol. 60, pp. 2194-2202, 2013

[73] D.B. Strukov et al., 'Prospects for Terabit-scale Nanoelectronic Memories', Nanotechnology, vol. 16, no. 1, pp. 137–148, 2005

[74] H.D. Lee et al., 'Integration of 4F2 selector-less crossbar array 2Mb ReRAM based on transition metal oxides for high density memory applications', Symposium on VLSI Technology, pp. 151-152, 2012

[75] N. Ramaswamy, 'Challenges in Engineering RRAM technology for high density applications', IEEE International Workshop On Integrated Reliability, pp. 1-5, 2012

[76] A. Flocke et al., 'Fundamental analysis of resistive nano-crossbars for the use in hybrid Nano/CMOS-memory', 33rd European Solid-State Circuits Conference, pp. 328-331, 2007

[77] H. Manem et al., 'Design considerations for variation tolerant multilevel cmos/nano memristor memory', in Symposium on Great Lakes Symposium on VLSI, pp. 287-292, 2010

[78] E. Linn et al., 'Complementary Resistive Switches for Passive Nanocrossbar Memories', Nature Materials, vol. 9, pp. 403-406, 2010

[79] J.J. Yang et al., 'Engineering nonlinearity into memristors for passive crossbar applications', Applied Physics Letters, 2012

[80] M. A. Zidan et al., 'Memristor-based memory: The sneak paths problem and solutions', Microelectronics Journal, 44 (2), pp. 176-183, 2013

[81] M. Qureshi et al., 'AC sense technique for memristor crossbar', Electronics Letters, vol. 48, pp. 757-758, 2012

[82] M. R. Stan et al., 'Molecular electronics: from devices and interconnect to circuits and architecture', Proceedings of the IEEE, vol. 91, pp. 1940-1957, 2003

[83] M. Liu et al., 'Application of nanojunction-based RRAM to reconfigurable IC', Micro Nano Letters, vol. 3, pp. 101-105, 2008

[84] L. Nielen et al., 'An Experimental Associative Capacitive Network based on Complementary Resistive Switches for Memory-intensive Computing', 2014 IEEE Silicon Nanoelectronics Workshop, 2014

[85] E. Lehtonen et al., 'Stateful Implication Logic with Memristors', IEEE/ACM International Symposium on Nanoscale Architectures, pp. 33-36, 2009

[86] A. Dehon, 'Nanowire-Based Programmable Architectures', ACM Journal on Emerging Technologies in Computing Systems, vol. 1, pp. 109-162, 2005

[87] K. K. Likharev et al., 'CMOL: Devices, Circuits, and Architectures', Introducing Molecular Electronics, vol. 680, pp. 447-477, 2006.

[88] S. Paul et al., 'Computing with Nanoscale Memory: Model and Architecture', pp. 1-6, 2009

[89] S. Paul et al., 'A Scalable Memory-Based Reconfigurable Computing Framework for Nanoscale Crossbar', IEEE Transactions on Nanotechnology, vol. 11, pp. 451-462, 2012

[90] K. Eshraghian et al., 'Memristor MOS Content Addressable Memory (MCAM): Hybrid Architecture for Future High Performance Search Engines', IEEE Transactions on VLSI Systems, vol. 19, pp. 1407-1417, 2011

[91] S.J. Lee et al., 'Complementary Resistive Switch (CRS) Based Smart Sensor Search Engine', 8th IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing, pp. 485 - 490, 2013

[92] G. Snider, 'Computing with hysteretic resistor crossbars', Applied Physics A, vol. 80, pp. 1165–1172, 2005

[93] E. Linn et al., 'Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations', Nanotechnology, vol. 23, pp. 305205/1-6, 2012

[94] L.O. Chua, 'Resistance switching memories are memristors', Applied Physics A, vol. 102, pp. 765-783, 2011

[95] H Aziza, et. al, 'A novel test structure for OxRRAM process variability evaluation', - Microelectronics Reliability, Vol. 53, N. 9, pp.1208-1212, 2013

7

CONCLUSION

This chapter summarizes the overall achievements of this dissertation and highlights some future research directions. Section 7.1 presents a summary of the main conclusions presented in this dissertation. Thereafter, Section 7.2 recommends future research directions.

7.1. SUMMARY

Chapter 1, "Introduction", briefly introduced resistive computing and the research focus of this thesis. It first described the motivation of this thesis. Thereafter, it overviewed the state-of-the-art of resistive computing with respects to the device, logic, architecture, compiler and application; it also highlighted the opportunities and challenges in each aspect. This thesis focused on addressing challenges with respect to logic (including primitive logic gates, interconnect design, circuit design and synthesis flow) and architecture (i.e., non von-Neumann architecture).

Chapter 2, "Background on Memristive Device and Its Potential", briefly introduced memristive device (including its brief history and working principle) and its potential when it is used in memories, logic circuits, and resistive computing paradigms (e.g., neuromorphic and computation-in-memory).

Chapter 3, "Primitive Logic Gate", first extended Snider logic (one of the logic design styles), then proposed scouting logic, and finally studied their resilience against device variability and other robust issues. First, this chapter explored the logic gate space of Snider logic. Second, this chapter presented a novel logic design style, called scouting logic. It executes a logic gate (i.e., AND, OR and XOR) by modifying standard read operations without changing their states. Hence, it does not impact the memristors' endurance. The sense amplifier used to implement logic functions is implemented using two styles (current and voltage based sense amplifiers). Third, the robustness of Snider logic and scouting logic were investigated. The robustness of Snider logic was analysed and mathematically formulated. Then, the impact of device variability, sneak path currents and parasitic resistance of nanowires on the robustness were investigated. For scouting logic, we discussed the impact of device variability on gate robustness and then proposed a methodology to counter it.

Chapter 4, "Interconnect Design", mainly explored how to efficiently connect individual gates. First of all, it explored three possible implementation methods of the interconnect network; they are (1) only using memristor crossbars, (2) using the CMOS peripheral circuits and (3) a hybrid of both the memristor crossbar and CMOS pass transistors. To illustrate the feasibility of such methods, a parallel adder is used as a case study. Among these three methods, the hybrid scheme performed best in terms of delay, energy and area. Second, it further explored how to use the memristor crossbar for both intra-tile and inter-tile (i.e., 2D bus) communication. In addition, a dedicated interconnect network was designed to address matrix transpose, which shows a new possibility to use the crossbar interconnect.

Chapter 5, "Circuit Design and Synthesis Flow", presented the implementation of ASICs and FPGAs using the knowledge obtained from Chapters 3 and 4. For ASICs, the circuits are first divided into look-up tables and then they are mapped on the memristor crossbar. Appropriate place-and-route schemes, to efficiently map the circuits on the crossbar, as well as several optimization schemes were proposed. In addition, a synthesis flow and performance estimation model were proposed. To realize FPGAs, we proposed two

novel implementations for the look-up tables utilizing memristor-based logic and their corresponding synthesis flow and performance estimation model.

Chapter 6, "Non-Von Neumann Architecture", proposed the Computation-In-Memory (CIM) architecture; it consists of an array of memristor devices that can be flexibly programmed to implement logic functions or memory and performs massive parallel operations. Therefore, it significantly reduces memory access and energy consumption as compared to conventional architectures (e.g., multi-core architectures). It explored the relevant data-intensive applications (e.g., parallel adder and DNA sequencing) to maximize the potential of CIM architecture.

7.2. FUTURE RESEARCH DIRECTIONS

Several recommendations are suggested to improve the state-of-the-art further. They are organized by the different research topics as listed below.

- **Primitive Logic Gate**

  1. Develop novel logic design styles. Innovative logic styles still need to be explored which utilize the unique properties of memristive devices, such as its multi-level resistance and non-volatility. These unique properties may lead to smaller and more energy-efficient logic circuits, or radically new ways of implementing logic circuits.
  2. Develop multi-functional logic gates. Existing logic styles can only implement a logic gate with a single function. If they are extended to multifunctional logic gates (e.g., AND-OR-NOT) like CMOS logic, it will simplify the circuits by using fewer gates.

- **Interconnect Design**

  1. Improve circuit-level interconnect. For VLSI circuits based on CMOS logic, typically interconnects become a bottleneck for large scale circuits in terms of both delay and power consumption. Therefore, the interconnection may become a bottleneck for memristor based logic circuits as well, when massive logic gates are integrated and the wire density increases. In addition, the routability of circuits should also be analyzed. All in all, more efforts should be paid to explore the interconnects for large scale circuits.
  2. Develop system-level interconnect schemes. Even though we explored a simple and naive bus communication infrastructure, the design methodology of system-level interconnect is not fully studied yet. For example, the implementation of more useful communication functions (e.g., scatter, gather) or the integration of a complex and powerful communication infrastructure (e.g., Network-on-Chip) need to be researched.

- **Circuit Design and Synthesis Flow**

  1. Develop novel arithmetic circuits. Memristor-based arithmetic circuits are still in an infancy stage and most research focuses on additions only. Even

though different approaches have been proposed to implement additions, their performance is still low as they need many steps to execute the addition. Therefore, speeding up arithmetic circuits is remaining an open question.

2. Simplify the CMOS controller. The CMOS controller is typically dominating the performance of the entire circuit. It is crucial to simplify the CMOS controller. This can lead to smaller area, lower delay and energy consumption.

- **Non-Von Neumann Architecture**

  Development of novel non-Von Neumann architectures. Although several pioneering architectures have been proposed (e.g., CIM [11], Pinatubo [15], AC-DIMM [44], the relevant research is still in its infancy stage. More novel architectures should be developed by utilizing the unique properties of the memristor technology, such as automata processors and neuromorphic processors.

REFERENCES

[1] S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, vol. 44, no. 1, pp. 31–38, 2011.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[3] "Exascale computing project," 2016. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends

[4] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb journal*, vol. 30, no. 3, pp. 202–210, 2005.

[5] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs *et al.*, "Memristor for computing: Myth or reality?" in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 722–731.

[6] D. E. Nikonov and I. A. Young, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2498–2533, 2013.

[7] X. Fu, M. Rol, C. Bultink *et al.*, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 813–825.

[8] G. Paun, G. Rozenberg, and A. Salomaa, *DNA computing: new computing paradigms*. Springer Science & Business Media, 2005.

[9] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[10] G. W. Burr, R. M. Shelby, A. Sebastian *et al.*, "Neuromorphic computing using non-volatile memory," *Advances in Physics: X*, vol. 2, no. 1, pp. 89–124, 2017.

[11] S. Hamdioui, L. Xie, H. A. D. Nguyen *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1718–1725.

[12] H. A. Du Nguyen, L. Xie, M. Taouil *et al.*, "Computation-in-memory based parallel adder," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 57–62.

[13] A. Haron, J. Yu, R. Nane *et al.*, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[14] P.-E. Gaillardon, L. Amarú, A. Siemon *et al.*, "The programmable logic-in-memory (plim) computer," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 427–432.

[15] S. Li, C. Xu, Q. Zou *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[16] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: avoiding the power wall with low-leakage, stt-mram based computing," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 371–382.

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

[18] H.-S. P. Wong, C. Ahn, J. Cao *et al.*, "Stanford memory trends," 2017. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends

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

[20] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[21] H.-S. P. Wong, H.-Y. Lee, S. Yu *et al.*, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[22] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, no. 6, pp. 1165–1172, 2005.

[23] J. Borghetti, G. S. Snider, P. J. Kuekes *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[24] S. Kvatinsky, D. Belousov, S. Liman *et al.*, "Magic–memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[25] L. Xie, H. A. D. Nguyen, M. Taouil *et al.*, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[26] L. Guckert and E. E. Swartzlander, "Mad gates—memristor logic design using driver circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 171–175, 2017.

[27] E. Linn, R. Rosezin, S. Tappertzhofen *et al.*, "Beyond von neumann? logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[28] T. You, Y. Shuai, W. Luo *et al.*, "Exploiting memristive bifeo3 bilayer structures for compact sequential logics," *Advanced Functional Materials*, vol. 24, no. 22, pp. 3357–3365, 2014.

[29] I. Vourkas and G. C. Sirakoulis, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Transactions on Nanotechnology*, vol. 11, no. 6, pp. 1151–1159, 2012.

[30] Y. Zha and J. Li, "Reconfigurable in-memory computing with resistive memory crossbar," in *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 1–8.

[31] L. Xie, H. A. D. Nguyen, J. Yu *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 335–340.

[32] J. Borghetti, Z. Li, J. Straznicky *et al.*, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proceedings of the National Academy of Sciences*, vol. 106, no. 6, pp. 1699–1703, 2009.

[33] S. Kvatinsky, N. Wald, G. Satat *et al.*, "Mrl-memristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012, pp. 1–6.

[34] G. S. Rose, J. Rajendran, H. Manem *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2033–2049, 2012.

[35] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115–119, 2013.

[36] Y. Levy, J. Bruck, Y. Cassuto *et al.*, "Logic operations in memory using a memristive akers array," *Microelectronics Journal*, vol. 45, no. 11, pp. 1429–1437, 2014.

[37] H. A. Du Nguyen, Y. Jintao, L. Xie *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 2017, pp. 57–64.

[38] M. Hu, J. P. Strachan, Z. Li *et al.*, "Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[39] K.-H. Kim, S. Gaba, D. Wheeler *et al.*, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, 2011.

[40] W. Kim, A. Chattopadhyay, A. Siemon *et al.*, "Multistate memristive tantalum oxide devices for ternary arithmetic," *Scientific reports*, vol. 6, 2016.

[41] S. Hamdioui, H. Aziza, and G. C. Sirakoulis, "Memristor based memories: Technology, design and test," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2014 9th IEEE International Conference On*. IEEE, 2014, pp. 1–7.

[42] I. Vourkas, D. Stathis, G. C. Sirakoulis *et al.*, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 206–217, 2016.

[43] Q. Guo, X. Guo, Y. Bai *et al.*, "A resistive tcam accelerator for data-intensive computing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 339–350.

[44] Q. Guo, X. Guo, R. Patel *et al.*, "Ac-dimm: associative computing with stt-mram," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 189–200, 2013.

[45] A. Chen, J. Hutchby, V. Zhirnov *et al.*, *Emerging nanoelectronic devices*. John Wiley & Sons, 2014.

[46] J. Cong and B. Xiao, "Fpga-rpi: A novel fpga architecture with rram-based programmable interconnects," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 864–877, 2014.

[47] L. Xie, H. A. Du Nguyen, M. Taouil *et al.*, "Non-volatile look-up table based fpga implementations," in *Design & Test Symposium (IDT), 2016 11th International*. IEEE, 2016, pp. 165–170.

[48] K. Ma, X. Li, S. Li *et al.*, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, no. 5, pp. 32–40, 2015.

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

[50] J. Yu, T. Hogervorst, and R. Nane, "A domain-specific language and compiler for computation-in-memory skeletons," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 71–76.

[51] J. Yu, R. Nane, I. Ashraf *et al.*, "Skeleton-based synthesis flow for computation-in-memory architectures," *IEEE Transactions on Emerging Topics in Computing, To appear*.

[52] L. Xie, H. A. Du Nguyen, M. Taouil *et al.*, "Boolean logic gate exploration for memristor crossbar," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2016, pp. 1–6.

[53] L. Xie, H. A. Du Nguyen, J. Yu *et al.*, "On the robustness of memristor based logic gates," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*. IEEE, 2017, pp. 158–163.

[54] H. Du Nguyen, L. Xie, J. Yu *et al.*, "Interconnect networks for resistive computing architectures," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017 12th International Conference on*. IEEE, 2017, pp. 1–6.

[55] L. Xie, H. A. Du Nguyen, M. Taouil *et al.*, "Interconnect networks for memristor crossbar," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 124–129.

[56] L. Xie, H. A. Du Nguyen, M. Taouil *et al.*, "A mapping methodology of boolean logic circuits on memristor crossbar," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 311–323, 2018.

[57] S. Kvatinsky, G. Satat, N. Wald *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.

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

[59] Intel xpoint memory. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/non-volatile-memory.html

[60] Crossbar 3d rram. [Online]. Available: http://www.crossbar-inc.com/

[61] M. Liu and W. Wang, "rfga: Cmos-nano hybrid fpga using rram components," in *NANOARCH*. IEEE, 2008.

[62] X. Tang et al., "A high-performance low-power near-vt rram-based fpga," in *FPT*. IEEE, 2014.

[63] V. Betz et al., *Architecture and CAD for deep-submicron FPGAs*. Springer Science & Business Media, 2012.

LIST OF PUBLICATIONS

International Journals

2. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Volume 37, Issue 2, Feb 2018.

1. H.A. Du Nguyen, **L. Xie**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *On the Implementation of Computation-In-Memory Parallel Adder*, IEEE Transactions on VLSI Systems (TVLSI), Volume 25, Issue 8, Aug 2017.

International Symposiums and Conferences

12. **L. Xie**, H.A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. Alfailakawi, S. Hamdioui, *Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing*,IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, July, 2017, pp. 151-156

11. **L. Xie**, H.A. Du Nguyen, J. Yu, M. Taouil, S. Hamdioui, *On the Robustness of Memristor Based Logic Gates*, IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Dresden, Germany, April, 2017, pp. 158-163

10. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, M Alfailakawi, *Non-Volatile Look-up Table Based FPGA Implementations*, IEEE International Symposium on Design and Test (IDT), Hammamet, Tunisia, December, 2016, pp. 165-170

9. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Boolean Logic Gate Exploration for Memristor Crossbar*, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Istanbul, Turkey, April, 2016, pp. 1-6

8. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Fast Boolean Logic Mapped on Memristor Crossbar*, IEEE International Conference on Computer Design (ICCD), New York City, USA, October, 2015, pp. 335-342. **Best Paper Award**

7. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Interconnect Networks for Memristor Crossbar*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, July, 2015, pp. 124-129

6. H.A. Du Nguyen, **L. Xie**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Interconnect Networks for Resistive Computing Architectures*, IEEE International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Dresden, Palma de Mallorca, Spain, April, 2017, pp. 1-6

5. H.A. Du Nguyen, **L. Xie**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Synthesizing HDL to memristor technology: A generic framework*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Beijing, China, July, 2016, pp. 43-48

4. H.A. Du Nguyen, **L. Xie**, R. Nane, M. Taouil, S. Hamdioui, K.L.M.Bertels, *Computation-In-Memory Based Parallel Adder*, IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Boston, USA, July, 2015, pp. 57-62

3. S. Hamdioui, **L. Xie**, H.A. Du Nguyen, M. Taouil, K.L.M. Bertels, *Memristor Based Computation-in-Memory Architecture for Data-Intensive Applications*, Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, March, 2015, pp. 1718-1725

2. H.A. Du Nguyen, J. Yu, **L. Xie**, M. Taouil, S. Hamdioui, D. Fey, *Memristive Devices for Computing: Beyond CMOS and Beyond von Neumann*, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Abu Dhabi, UAE, October, 2017, pp. 1-10

1. S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, **L. Xie**, N. Wald, S. Joshi, H. Elsayed, H. Corporaal, K.L.M. Bertels, *Memristor For Computing: Myth or Reality?*, Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, March, 2017, pp. 722-731

Workshops

12. **L. Xie**, H.A. Du Nguyen, J. Yu, M. Taouil, S. Hamdioui, *FPGA Implementations Based on Memristor Logic Circuits*, ICT.OPEN, Amersfoort, Netherlands, March, 2017

11. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Non-Volatile Look-up Table Based FPGA Implementations*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Stockholm, Sweden, January, 2017

10. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui and K.L.M. Bertels, *Boolean Logic and Interconnect for Memristor Crossbar*, ICT.OPEN, Amersfoort, Netherlands, March, 2016

9. **L. Xie**, H.A. Du Nguyen, M. Taouil, S. Hamdioui, K.L.M. Bertels, *Memristor Crossbar Based Logic and Interconnect Design*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Prague, Czech, January, 2016

8. H.A. Du Nguyen, **L. Xie**, M. Taouil, S. Hamdioui, K.L.M. Bertels, *A Synthesis Framework for Memristor Crossbar*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Stockholm, Sweden, January, 2017

7. H.A. Du Nguyen, **L. Xie**, M. Taouil, R. Nane, S. Hamdioui, K.L.M. Bertels, *CIM Based Parallel Adder Implementations and Evaluations*, Workshop on Memristor Technology, Design, Automation and Computing (MDAC) in conjunction with HiPEAC, Prague, Czech, January, 2016

6. H.A. Du Nguyen, **L. Xie**, M. Taouil, R. Nane, S. Hamdioui and K.L.M. Bertels, *CIM Architecture Communication Schemes*, International Workshop on In-Memory and In-Storage Computing with Emerging Technologies (IMISCET) in conjunction with PACT, Haifa Israel, September, 2016

5. H.A. Du Nguyen, **L. Xie**, M. Taouil, R. Nane, S. Hamdioui and K.L.M. Bertels, *CIM Based Parallel Adder Implementations and Evaluations*, ICT.OPEN, Amersfoort, Netherlands, March, 2016

4. D. Wouters, A. Siemon, **L. Xie**, S. Menzel, R. Waser, S. Hamdioui, *Influence of ReRAM Device Characteristics on the Performance of Logic-in-Memory Concepts*, International Conference on Memristive Materials, Devices & Systems (MEMRISYS), Athens, Greece, April, 2017

3. J. Yu, H.A. Du Nguyen, **L. Xie**, M. Taouil, R. Nane, S. Hamdioui, K.L.M. Bertels, *FPGA Implementations Based on Memristor Logic Circuits*, ICT.OPEN, Amersfoort, Netherlands, March, 2017

2. S. Hamdioui, M. Taouil, H.A. Du Nguyen, M.A.B. Haron, **L. Xie**, K.L.M. Bertels, *CIMx: Computation in-Memory Architecture Based on Resistive Devices*, International Workshop on Cellular Nanoscale Networks and their Applications (CNNA), Dresden, Germany, August, 2016

1. S. Hamdioui, M. Taouil, H.A. Du Nguyen, M.A.B. Haron, **L. Xie**, K.L.M. Bertels, *Memristor: The Enabler of Computation-in-Memory Architecture for Big-Data*, International Conference on Memristive Materials, Devices & Systems (MEMRISYS), Paphos, Cyprus, November, 2015

CURRICULUM VITÆ

Lei XIE

L. Xie was born 1987 in Yinchuan, China. He received his B.Sc. degree in microelectronics in 2010 from Xi'an Jiaotong University, Xi'an, China. After that he received M.Sc degree in Microelectronics and Solid State Electronics from the same university. In September 2013, he joined the Department of Quantum and Computer Engineering at the Faculty of Electrical Engineering, Mathematics, Computer Science in Delft University of Technology. His research interests include memristor-based logic circuit design for both ASIC and FPGA.