Memristor-Based Circuits and Architectures

Shahar Kvatinsky

Memristor-Based Circuits and Architectures

Research Thesis

In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Shahar Kvatinsky

Submitted to the Senate of the
Technion – Israel Institute of Technology

Sivan, 5774 Haifa May 2014

This research thesis was done under the supervision of Prof. Avinoam Kolodny, Prof. Eby Friedman, and Prof. Uri Weiser in the department of Electrical Engineering.

The generous financial help of Hasso Plattner Research Institute, Irwin and Joan Jacobs, and Andrew and Erna Finci Viterbi is gratefully acknowledged.

Table of Contents

Table of Figures

Chapter 3.2.1

Chapter 3.2.2

Chapter 3.2.3

Chapter 3.2.4

Chapter 3.2.5

Chapter 3.2.6

Chapter 3.3.1

Chapter 3.3.2

Chapter 3.3.3

Table of Tables

Chapter 3.2.4

Chapter 3.2.5

Chapter 3.2.6

Chapter 3.3.2

Chapter 3.3.3

Abstract

Advancements in computer capabilities in the last fifty years had been closely linked to miniaturization of CMOS technology, while the underlying structure of digital computing systems has been based on von Neumann architecture, where the memory and execution units are logically and physically separated, using various types of interconnect for communication between them. Recently, device scaling has slowed down, while electrical interconnect has become both a performance bottleneck and a major source of power dissipation, which is currently the most critical limiter for technology growth. Conventional memory technologies, such as Flash, DRAM, and SRAM, are unable to keep up with market requirements for higher density and lower power. Flash memory has already achieved its physical limits, and cannot be scaled further, primarily due to its limited endurance.

These problems can be addressed by emerging new semiconductor devices, such as memristors, which are useful both as memory cells and as novel switching circuits which can be used to augment traditional CMOS gates. Memristors are simple two-terminal resistors, where the resistance can be changed by the electrical current. The resistance serves as a stored variable. Memristors can also be interconnected to perform Boolean operations. Since memristors can be fabricated in high density at the intersection of nanoscale width metal lines, common to all silicon circuits and located on top of the silicon layer, these new devices hold promise to provide continued growth in functional density. The primary focus of this thesis is on architecturally integrating memory with computational capabilities, based on exploiting these new devices. These memristor-based structures will greatly enhance the speed and power of digital computing beyond Moore scaling, while maintaining compatibility with standard CMOS technology. From an architectural viewpoint, memristor-based circuits will lead to innovative *memory-intensive computing structures and systems*.

The focus of this research is on developing memristor-based applications at the circuit and architecture levels. Memristors are investigated from the point of view of circuit designer and computer architect, including describing the desired device for different applications and modeling a general memristor model – TEAM – to fit different memristive technologies. The TEAM model is simple (*i.e.*, requires low

computational effort) and sufficiently accurate. The TEAM model is implemented in VerilogA to be used in SPICE simulations.

Various logic circuits with memristors have been proposed and design methodologies for them are developed. IMPLY (material implication), MAGIC (Memristor Ratioed Logic), and Akers logic arrays are logic families that can be performed within memristive memories, enabling in-memory computing. MRL (Memristor Ratioed Logic) is a different logic family used for hybrid CMOS-memristor logic gates to increase the logic density and extend Moore's law.

Additionally, the multistate register, a novel memory structure that stores multiple values within a single register, is proposed. A multistate register is designed based on an RRAM crosspoint array on top of a CMOS register with a relatively low area. The area of a single state in a 64 state RRAM multistate register is only 1.3% of a stored state in CMOS register. The multistate register is embedded within CPU pipelines to enable new memory intensive architectures, such as Continuous Flow Multithreading (CFMT). CFMT is a multithreaded processor that is as simple as Switch on Event Multithreading (SoE MT) with high performance and low energy. CFMT is designed and implemented with an FPGA, presenting a performance improvement of 32% on average with an energy reduction of 8.5%, as compared to SoE MT.

Abbreviations

CFMT	Continuous Flow Multithreading
CRS	Complementary Resistive Switching
DRAM	Dynamic Random Access Memory
FPGA	Field Programmable Array
IPC	Instruction per Cycle
MAGIC	Memristor Aided Logic
MIM	Metal-Insulator-Metal
MPR	Multistate Pipeline Register
MRL	Memristor Ratioed Logic
NVM	Nonvolatile Memory
PCM	Phase Change Memory
RRAM	Resistive Random Access Memory
SCM	Storage Class Memory
SMT	Simultaneous Multithreading
SoE MT	Switch-on-Event Mutlithreading
STDP	Spike Timing Dependent Plasticity
STT MRAM	Spin-Transfer Magnetoresistance Random Access Memory
TEAM	Threshold Adaptive Memristor
VLSI	Very Large Scale Integration

Chapter 1 Introduction

For almost fifty years, integrated electronic circuits built with semiconductor devices have provided significant growth in the number of processing elements and memory bits available to system developers. This growth has provided orders of magnitude improvements in speed, power consumption, and reliability, together with significant reductions in the cost per device. These trends are direct consequences of frequent miniaturization of device dimensions in the semiconductor fabrication process, as originally described by Gordon Moore in 1965, predicting the growth and proliferation of digital computing and its applications ("Moore's Law"). Throughout this era, the underlying structure of digital computing systems has been based on the classical stored program machine architecture described by von Neumann in the 1940's, which is characterized by a separation between functional units for instruction execution and data/instruction storage ("von Neumann architecture").

Moore's law, however, cannot be sustained indefinitely. There is broad agreement that nanoscale CMOS transistor sizes will reach fundamental physical limits within the next decade [1]. Even before the eventual ending of Moore's law due to technological limitations, the field of computing is already struggling with other fundamental problems which require innovative solutions. The first problem is related to the delay and bandwidth required to access memory, and is popularly known as "the memory wall" [2]. Another problem is the power crisis related to energy dissipation in computers [3]. These challenging issues are currently perceived as major disruptions on the evolutionary path of computing, calling for significant investments in research to develop new structures for next generation computing systems.

In future years, when device sizes will no longer be scalable, microelectronic technology will need innovations "beyond Moore" to support novel applications. These enhancements may include revolutionary new emerging devices such as carbon nanotubes or spintronic devices. A less radical hybrid approach, combining standard CMOS with new technologies, is expected to provide a more practical and immediate growth path over the next 20 to 30 years. An example of a "more than Moore" technology is multi-layered integrated circuits (*e.g.*, three-dimensional circuits [4])

which are becoming commercially available. Other new technologies which will extend the capabilities of CMOS are memristive devices. This thesis focuses on memristive technologies and their impact on computers.

1.1 Memristors

Over the past 25 years, flash memory based on charge trapping in MOS transistors has been scaled aggressively, even exceeding Moore's law. Scaling below 20 nm involves formidable challenges, particularly an increase in bit error rate and a reduction in write endurance (the number of write cycles before the memory becomes unreliable). These challenges become intolerable, when flash process technology scales below 15 nm [5]. In recent years, many alternative technologies have been explored to find a replacement for flash. For most of these candidate technologies, the stored data is represented as a resistance and the storage device is fabricated within the metal layers. These technologies share similar properties – nonvolatility, relatively high write endurance, high density, good scalability below 10 nm, and fast read and write. Certain emerging memory technologies have sufficient speed and endurance to be considered as SRAM and DRAM replacements as well, enabling the use of universal memory [6]. These emerging nonvolatile memory technologies can be considered as *memristors*, or more precisely as *memristive devices*.

1.1.1 The Theory of Memristors

In 1971, Leon Chua conceived the need for an additional fundamental circuit component in addition to the resistor, capacitor, and inductor [7]. Chua reasoned the existence of a missing circuit element from symmetry reasons, looking at the six possible combinations of the relationships of four fundamental circuit variables - the voltage V, current I, flux φ, and electric charge q. While the charge is the integral upon time of the current and the flux is integral upon time of the voltage, the other possible relationships are connected by two-terminal circuit components. Resistors connect voltage to current by Ohm's law ($V = IR$), capacitors connect charge to voltage ($q = CV$), and inductors connect current to flux ($\varphi = LI$). The sixth possible relationship is the connection between charge and flux and is not covered by any

conventional circuit element. Chua reasoned, for the sake of completeness, the existence of a fourth fundamental circuit element that connects the charge and flux and named the device the *memristor*, as a short for '*memory resistor*'. The six combinations of the relationships are illustrated in Figure 1.

Formally, a charge-controlled memristor is given by

$$v(t) = M\big(q(t)\big)i(t), \tag{1}$$

where

$$M\big(q(t)\big) \equiv d\varphi(q)/dq. \tag{2}$$

Similarly, a flux-controlled memristor is given by

$$i(t) = W\big(\varphi(t)\big)v(t), \tag{3}$$

where

$$W\big(\varphi(t)\big) \equiv dq(\varphi)/d\varphi. \tag{4}$$

The memristance of the memristor $M\big(q(t)\big)$ has the units of resistance (and the units of $W\big(\varphi(t)\big)$ are of conductance) and depends on the integration of the current passed through the device upon time. The memristor is therefore actually a passive two-port element with variable resistance, which changes upon the history of the device (*i.e.*, the memristance depends on the total charge passed through the device).

In 1976, the theory of memristors was extended by Chua and Kang to a nonlinear dynamical system named memristive systems [8]. Similarly to memristors, a memristive device is a passive two-terminal device with varying resistance. The difference between a memristor and memristive device is how the resistance changes. In memristive devices, the resistance depends on an internal state $x \in \Re^n$, which depends on the history of the device (in terms of the past current passed through the device, or, alternatively, the past voltage across the device) and not directly on the charge or flux.

Formally, a current-controlled time-invariant memristive device is represented by

$$v(t) = M(x, i)i(t), \tag{5}$$

$$dx/dt = f(x, i), \tag{6}$$

where $M(x, i)$ is the memristance of the device. Similarly, a voltage-controlled time-invariant memristive device is given by

$$i(t) = W(x, i)v(t), \tag{7}$$

$$dx/dt = f(x, v). \tag{8}$$

Memristors and memristive devices exhibit hysteresis in their current-voltage curve.

Although the shape of the hysteresis varies for different devices, it is always passes through the origin. The hysteresis depends on the input, where for high input frequencies the device behaves as a linear resistor. An example of possible I-V curve is shown in Figure 2. Note that memristors are a private case of memristive devices, where $f(x, i) = i$.

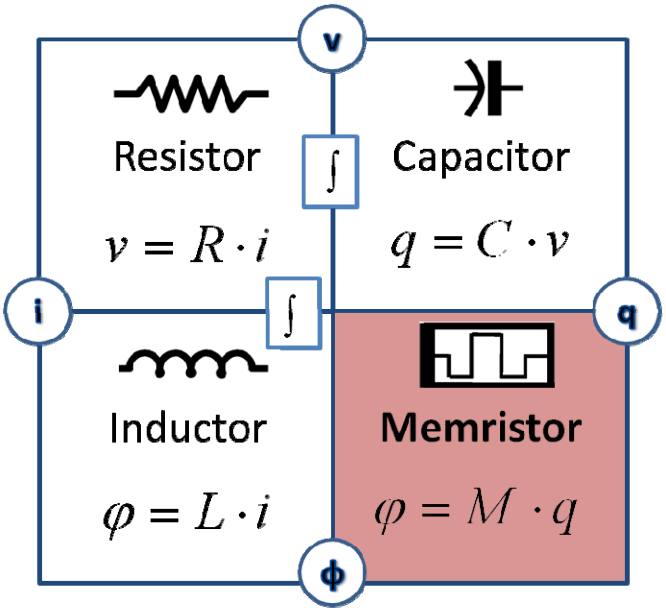


Figure 1. Illustration of the six combinations of the relationships between voltage *v*, charge *q*, flux ϕ, and current *i*. The memristor connects the charge and flux.

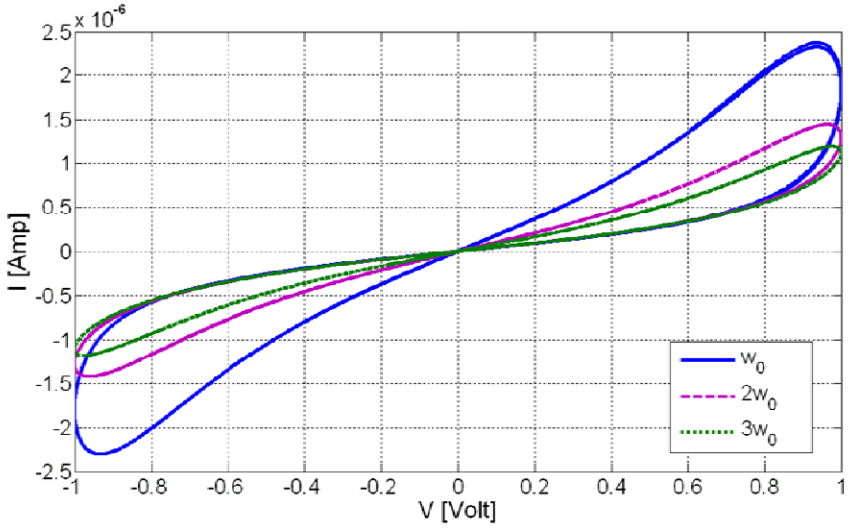


Figure 2. Example of a current-voltage curve of memristors and memristive devices for different input frequencies.

1.1.2 Practical Memristors

While Chua and Kang presented physical memristive devices in [8] (*e.g.*, ionic systems and discharge tubes), the topic of memristors and memristive devices have not attracted much attention for more than 35 years. In 2008, Hewlett Packard connected the theory of memristive devices to TiO_2 resistive switches [9]. Initially, Hewlett Packard claimed that their device is similar to ideal memristors, and proposed a model for the structure and behavior of their devices. The proposed structure is shown in Figure 3 and the proposed representation of (5) and (6) is

$$M(x,i) = R_{ON}\frac{x(t)}{D} + R_{OFF}(1 - \frac{x(t)}{D}), \tag{9}$$

$$f(x,i) = \mu_V \frac{R_{ON}}{D} i(t), \tag{10}$$

where R_{ON} is the resistance when $x(t) = D$, and R_{OFF} is the resistance when $x(t) = 0$. The state variable $x(t)$ is limited to the physical dimensions of the device, *i.e.*, the value is within the interval $[0, D]$.

Although the model proposed by Hewlett Packard is elegant, it does not match real devices, including their own TiO_2 device. Other models that better fit real devices have been proposed, as comprehensively explained in Chapter 3.1. The announcement of Hewlett Packard, however, sparked an interest in memristors and memristive systems. Additional resistive memory devices, other than TiO2 resistive switches, such as different resistive switches and spin-transfer torque magnetoresistive random access memory (STT-MRAM) have been redescribed in terms of memristive systems [10-14].

All of the different devices that can be considered as memristive devices share several characteristics: they are fabricated as oxides sandwiched between two metals (metal-insulator-metal structure, also named MIM), and their size is relatively small (for most devices it is the minimum feature size of the technology). Additionally, as described by the definition of memristive devices, these devices have varying resistance and are nonvolatile (*i.e.*, no voltage is applied to retain the resistance). Due to these characteristics and their relatively low switching time (from sub-nanoseconds to tens of nanoseconds), high endurance (the number of write cycles before the memory becomes unreliable, typically from 10^9 to 10^{15}), and low switching energy (typically 0.1 to 1 pJ), memristive technologies are primarily investigated as memory applications and considered as *emerging nonvolatile memory technologies*. It is

common referring to all (or some) of these technologies as Resistive RAM (RRAM or ReRAM).

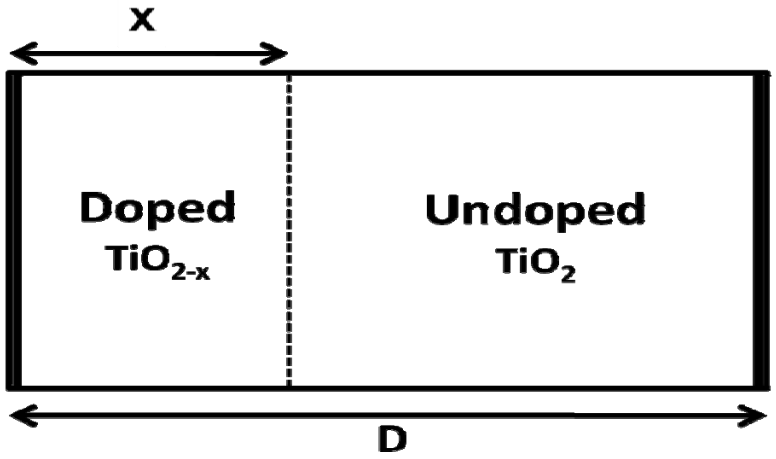


Figure 3. Hewlett Packard original device model. The device is composed of two regions: doped and undoped. The total resistance of the device is the sum of the resistances of both regions, as described by (9).

1.2 Memristor-Based Applications

While the semiconductor industry focuses on the use of memristive technologies as a replacement for existing memory technologies (*i.e.*, Flash, DRAM, and SRAM), these devices can be used for many other applications as well. Memristors are used in hardware neural networks, both to implement different learning algorithms (*e.g.*, STDP [15-17] and back propagation [18]) and neuromorphic systems (hardware that mimic the brain) [19]. Memristors can also been used in analog circuits, for example as reconfigurable resistors to change the properties of the circuit [20]. Another interesting application is the use of memristors as part of logic circuit, as comprehensively explained in Chapters 1.2.2 and 3.2.

Looking at computer architectures, memristors can be an enabler to a new and disruptive era in computer architecture – the era of memory intensive computing, where the computation engine is integrated with numerous memory devices (*i.e.*, memristors). The straightforward way is to use memristors as improved replacements to existing memory technologies and benefit from the improve characteristics of the replacements - higher density, no leakage, high endurance, *etc.* Using these technologies as SRAM replacements will significantly increase on-die memory. Alternatively, additional memory levels can be added to the memory hierarchy. For

example, Sony and Micron plan to commercialize RRAM as a Storage Class Memory (SCM), a memory hierarchy between DRAM and flash [21]. SCM requires nonvolatility and high density, as well as high performance. Adding more cache levels is another example. Memristors are, however, much more than just having replacements to existing memory technologies.

1.2.1 Memory Intensive Architectures

Memristors add new characteristics to existing memory technologies. For example, the levels of volatile memory (*i.e.*, register file, cache, and main memory) become nonvolatile. Memristors add fast, dense, and nonvolatile memory that is located on-top of the logic gates. Memristors therefore can be used in a different manner than random access memories. The memristors can be used to integrate memory and logic, enabling memory intensive architectures, where processors are abundant with non-volatile, fast memory. This memory is used to enhance performance and decrease energy.

The small size of the memory devices and the possibility to stack several layers of memory, one on top of the other, can significantly increase the capacity of the memory, including cache hierarchies, while leakage power is lower. The additional memory can also be used not only for conventional caches (*i.e.*, data and instruction cache, private and shared cache, *etc.*) but also for new cache architectures, including specific purposes caches. Examples to this can be using different caches for different threads, or alternatively, different caches for specific content (*e.g.*, floating point, SIMD). Another example is to use memristors to implement NAHALAL-like cache systems [22], where the private caches are located on-top of the CPUs in the memristor layer.

Furthermore, the additional memory can be used to increase the capacity of other elements within the processor, such as branch predictors, instruction queues, prefetching structures, reorder buffers, and other buffers. The increase in the capacity increases predictions and speculations of the processor and therefore trades off the power consumption – although storing data within memristor-based structures is low power, the increased speculation consume more power.

The additional memory elements can also be used to store data, which is typically not stored due to the limitations of conventional technologies. For example, it is

possible to store the results of previously executed instructions to perform instruction reuse and have hardware memoization [23]. It is also possible to have many checkpoints within the processor. Another example is CFMT, as presented in Chapter 3.3, where the states of different instructions for multiple threads are stored to enhance performance of multithreading processors.

1.2.2 Logic with Memristors

One interesting application of memristive circuits is to perform logic operations. With memristive logic gates, novel memory intensive architectures can be developed, including non-von Neumann architectures. The use of memristors to perform logical operations has been proposed in several different ways. In some logic families, memristors are integrated with CMOS structures to perform the logical operation, while the logical values are represented by voltage levels. Memristors can be used as reconfigurable switches for FPGA-like architectures [24-25] or as computational elements within logic gates [26].

Another approach for logic with memristors is to treat resistance as the logical state, where the high and low resistance are considered, respectively, as logical zero and one. For this approach, the memristors are the primary building blocks of the logic gate. Each memristor acts as an input, output, computational logic element, and latch in different stages of the computing process [27]. This approach is suitable for crossbar array architectures and can therefore be integrated within a standard memristor-based crossbar, commonly used for memory. This approach is appealing since it provides an opportunity to explore advanced computer architectures different from the classical von Neumann architecture. In these architectures, the memory can perform logic operations on the same devices that store data, *i.e.*, performing computation inside the memory. Material implication (IMPLY logic gate) [28] is the basic logical element using this approach, combining state memory and a Boolean operator. Additional logic families, which extends the IMPLY logic gate by using certain variations of a regular memristor-based crossbar, have also been proposed [29-30]. A schematic of the IMPLY logic gate is shown in Figure 4.

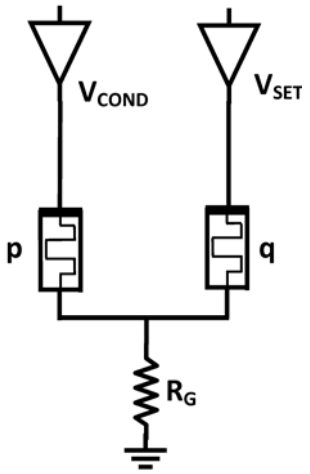


Figure 4. Schematic of an IMPLY logic gate. The gate consists of two memristors *p* and *q*, and a resistor R_G.

1.3 Research Goals and Methods

In this research, the capabilities and limitations of memristors are studied and analyzed from single device level for better understanding of the capabilities of circuits and architectures with memristors. Models of memristors are developed for different memristor technologies to be used in circuit simulations.

The integration of memristors with CMOS is explored for memory and logic circuits. Different logic gates are proposed, both for non-von Neumann architectures, where memristive memories have also computing capabilities (in-memory computing), and for hybrid CMOS-memristor logic gates for a beyond Moore approach. Memory circuits are designed for new uses, different than the conventional memory hierarchy. These memory circuits are integrated with processors to open a path for novel memory intensive architectures.

This research is done in different abstraction levels: device level, circuit level, and architecture level. For device level the properties of memristors are investigated and characterized, and device models are developed to be used in circuit simulations. In the circuit level, different digital and analog circuits are designed for logic and memory applications. Design methodologies are developed for proper circuit design, including procedures to select the exact device and circuit parameters. Memristive circuits are used to develop memory intensive architectures.

This research is multidisciplinary, varying from device physics, VLSI physical design, VLSI circuit design, electronic design automation, and computer architecture. We also combined knowledge from information theory and machine learning.

For memristor model development, we use Matlab to evaluate and investigate the model and compare it to other memristor models. The model is implemented in Verilog-A and embedded in SPICE. We develop design methodologies for circuits, and design them in SPICE, using Cadence Virtuoso. Architectures are evaluated in a cycle accurate in-house simulator for performance evaluation, and by CACTI and McPAT for energy evaluation. For the design of memory architectures, we also use NVSim. For a proof of concept, the CFMT architecture is also implemented in real hardware (Xilinx Virtex 6), using Verilog and ModelSim for verification.

1.4 Thesis Structure

This thesis is organized as a collection of papers published in scientific journals and refereed conference proceedings. All of the papers describe research results obtained during this PhD study. Chapter 2 provides a short overview of the main contributions of this thesis. The published papers are organized in Chapter 3 in three subsections according to the main parts of the research: device characteristics and modeling, logic circuits, and computing system architecture. In Chapter 4, the main results are summarized, some additional research topics are described, and suggestions for future research are discussed.

Chapter 2 Summary of Contributions

In this chapter, our contributions in the field of memristor-based circuits and architectures are summarized. Our main observations and solutions to the different research problems considered in this thesis are outlined. For convenience, each subchapter is devoted to a different aspect of this research.

2.1 Device Level

2.2.1 Device Characterization

Memristors have many different faces, from the theoretical devices envisioned by Chua in 1971, to the extended theory of memristive device, to the numerous different resistive technologies that have emerged in recent years. Since memristive technologies are currently immature, standardization of the characteristics of memristors remains to be done. The desired characteristics differ for diverse applications.

In this research, we study and analyze the capabilities and limitations of different memristive technologies, and define the desired characteristics of memristors for different applications from the viewpoint of an integrated circuit designer. Understanding the desired characteristics for different applications can assist device and material engineers in providing the appropriate behavior when developing memristive devices, thereby optimizing these devices for different applications. Further details on the desired memristor are found in Chapter 3.1 and [31].

2.2.2 Device Modeling

Several models for memristive devices have been developed. One type of models is physical models that try to fit the dynamic behavior of a specific memristor [32-35]. Usually, physical models are complicated and based on mathematical fitting of experimental results for a specific device under a certain experimental set, while the actual physical mechanism is still unknown. A different approach for memristor modeling is to define mathematical models, obeying the theory of memristors, without a connection to practical devices [36-39]. Usually, mathematical models are similar to Chua's original definition and cannot predict the behavior of real devices.

In this research, a general mathematical model - TEAM, ThrEshold Adaptive Memristor model – is developed. The TEAM model is flexible and can be fit to any

practical memristive device. As shown in this thesis, the TEAM model is reasonably accurate and computationally efficient, and is more appropriate for circuit simulation than previously published models. The TEAM model is implemented in Verilog-A, and is widely used in SPICE simulations. Further details on the TEAM model are found in Chapter 3.1 and [40-41].

2.2 Logic Circuits

2.2.1 Material Implication (IMPLY)

Although IMPLY gates have been fabricated and proved to work [28], the design issues of these logic gates have not been discussed. Additionally, the design of complete combinatorial system based on IMPLY gates is not trivial. In this research, the behavior of IMPLY logic gates is analyzed and evaluated, and the tradeoff between speed and correct logic behavior is described. An approximate analytic model to evaluate the speed of the circuit and the internal state drift of the memristors is proposed. We develop a methodology for designing IMPLY logic family, based on a general design flow, suitable for all deterministic memristive logic families, and includes some additional design constraints to support the IMPLY logic family. The design flow is shown in Figure 5. Additionally, techniques for performing logic within memristive crossbars based on IMPLY logic gates are discussed and proposed. Further details on IMPLY logic design and IMPLY within the memory are found in Chapter 3.2 and [42-43].

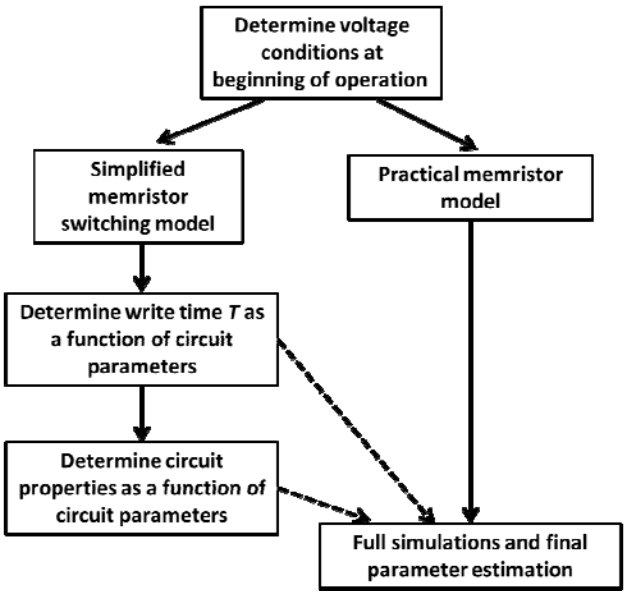


Figure 5. Proposed design flow for IMPLY logic.

2.2.2 MAGIC

IMPLY gates require different voltage levels within the circuit and additional circuit components (for example, a controller and an additional resistor within each row of the crossbar), dissipates high power, has high computational complexity, and requires complicated control circuitry. Additionally, the result is stored by one of the inputs and not a dedicated output memristor.

We propose a different memristive-only logic family, Memristor Aided LoGIC (MAGIC) that overcomes the disadvantages of IMPLY. MAGIC does not require a complicated structure and enables stable evaluation of the gate function. Stable evaluation is achieved by applying a single voltage pulse at the gateway of the circuit. MAGIC NOR gates can also be fabricated within a crossbar, enabling computing within memory. The schematic of MAGIC NOR is shown in Figure 6. Further details on IMPLY logic design and IMPLY within the memory are found in Chapter 3.2 and [44].

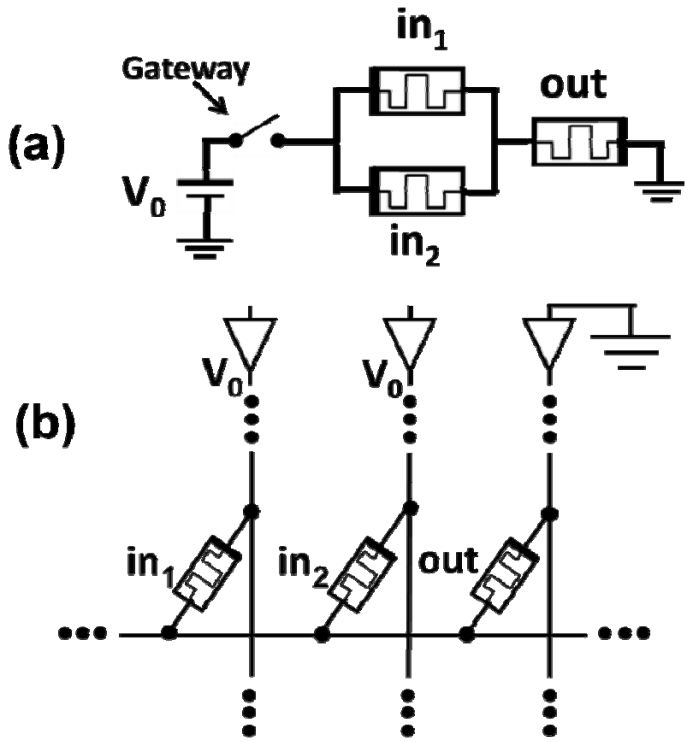


Figure 6. MAGIC NOR. (a) Basic schematic, the gate consists of two input memristors *in*$_1$ and *in*$_2$, and an output memristor *out*, and (b) a MAGIC NOR gate within a memristive crossbar.

2.2.3 MRL

We propose MRL (Memristor Ratioed Logic), a hybrid CMOS-memristor logic family, which increases the logic density. In MRL, OR and AND logic gates are based on memristors, and CMOS inverters are added to provide a complete logic structure and signal restoration. The MRL family is compatible with standard CMOS logic since the logical state is represented by voltage as in CMOS. We develop an analytic model to evaluate the speed of the circuit and discuss design issues and considerations, including area and power. The schematic of an MRL NAND and NOR are shown in Figure 7. Further details on MRL gates are found in Chapter 3.2 and [45-46].

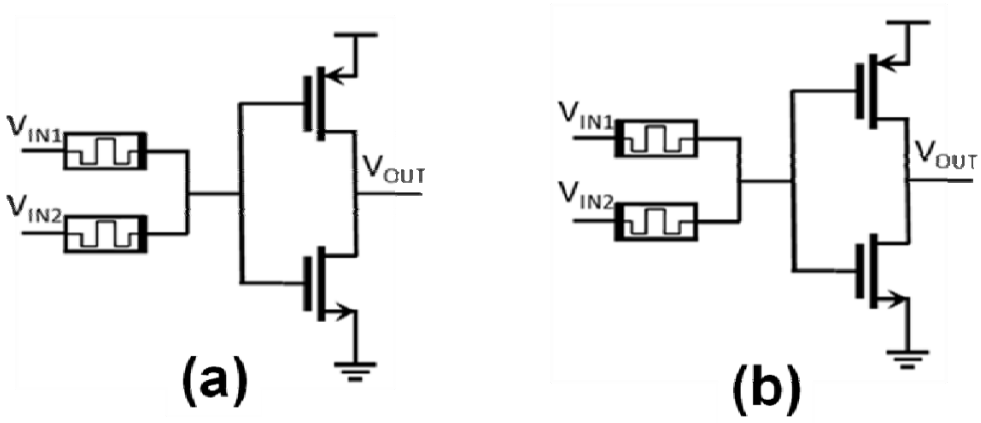


Figure 7. MRL gates. (a) A two-input MRL NAND and a (b) two-input MRL NOR.

2.2.4 Akers Logic Arrays

In 1972, Sheldon Akers proposed a theoretical logic array that supports the execution of any Boolean function by flowing data across an array of primitive logic cells. Since the benefit of an Akers logic array with conventional semiconductor technology (*i.e.*, CMOS technology) is limited, Akers array has been treated as a mathematical concept without implementing it in real hardware. In this research, the theory by Akers is used to design a memristive Akers logic array that support in-memory computation. We show that the proposed logic array can be used in a modified CRS memory array, combining logic operations and memory. We demonstrate Boolean operations such as XOR and sorting of bits. The schematic of a memristive Akers logic array is shown in Figure 8. Further details on memristive Akers logic arrays are found in Chapter 3.2 and [47].

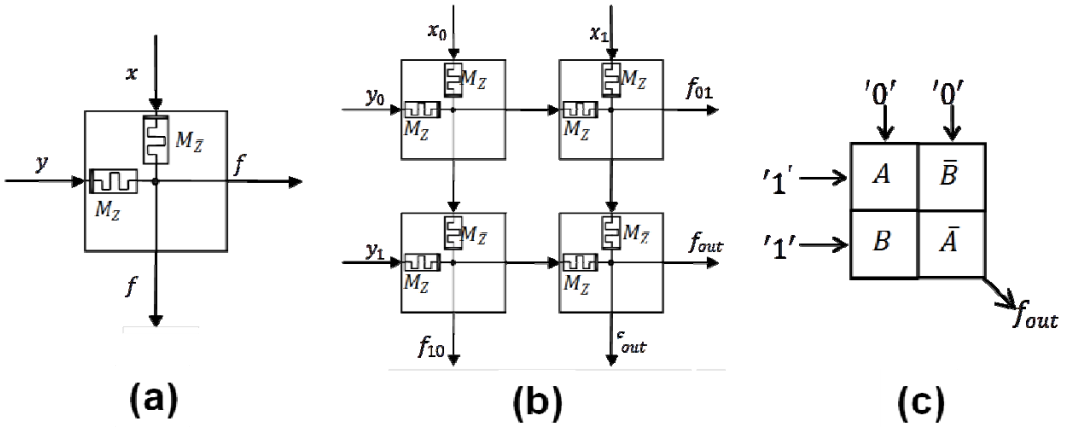


Figure 8. Memristive Akers logic array. (a) Primitive logic cell, consists of two memristors M_Z and $M_{\bar{Z}}$. The inputs of the primitive logic cell are two voltages x and y, and the initial resistance of M_Z. (b) A two by two memristive Akers array and (c) a two input XOR, where $f_{out} = XOR(A, B)$.

2.3 Multistate Registers and Its Implications

2.2.1 Multistate Register

Storing data is the primarily application of memristors, usually for replacing conventional memory technologies in standard memory structures within the memory hierarchy. A different approach is considered in this research, where a novel memory structure, the multistate register, is proposed and designed. The multistate register is used to store multiple data bits, where only a single bit is active and the remaining data bits are idle. The active bit is stored within a CMOS flip flop, while the idle bits are stored in a memristive crossbar co-located with the flip flop. Multistate registers open opportunity for new applications and architectures, exploiting the density and low power of memristors. The schematic of an RRAM-based multistate register is shown in Figure 9. Further details on multistate registers are found in Chapter 3.3 and [48].

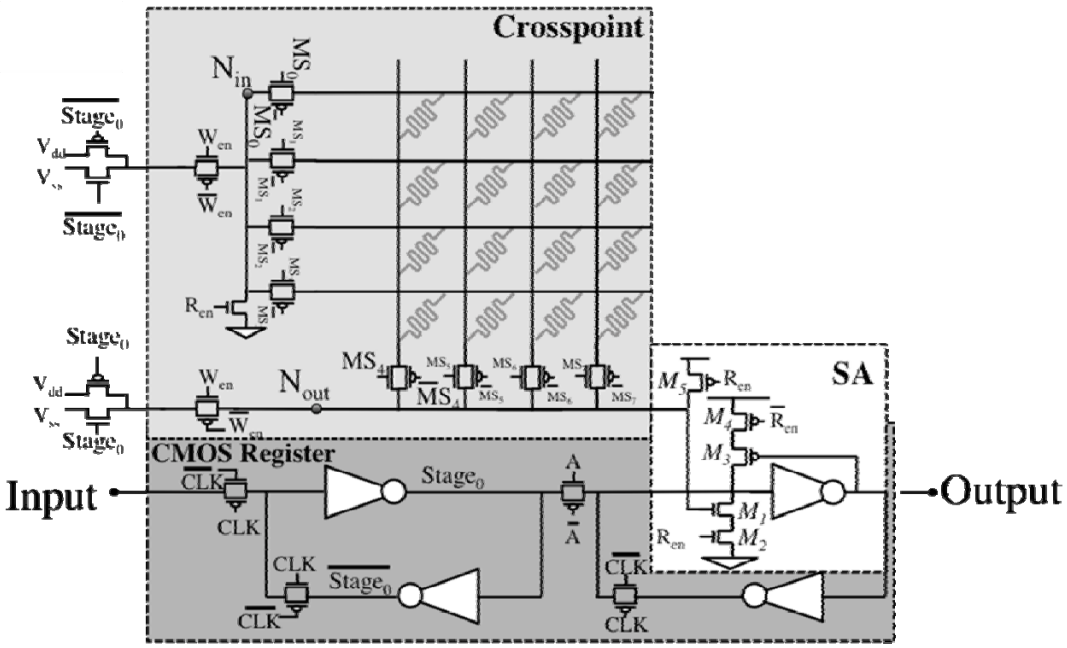


Figure 9. Schematic of an RRAM 16 states multistate register. The multistate register consists of an RRAM crosspoint on top of a CMOS D flip flop.

2.2.2 Continuous Flow Multithreading

The use of multistate registers to store the microarchitectural state of multiple threads within the processor pipeline is proposed. We call this use a multistate pipeline register (MPR). Using MPRs can eliminate the need to flush the pipeline upon a thread switch in Switch-on-Event (SoE) multi-threading machines. We call the new microarchitectural scheme, Continuous Flow Multi-Threading (CFMT), and compare the performance and power consumption against traditional SoE machines. Memristor-based CFMT significantly improves performance as compared to SoE, while reducing energy. A CFMT processor is illustrated in Figure 10. Further details on CFMT are found in Chapter 3.3 and [49-50].

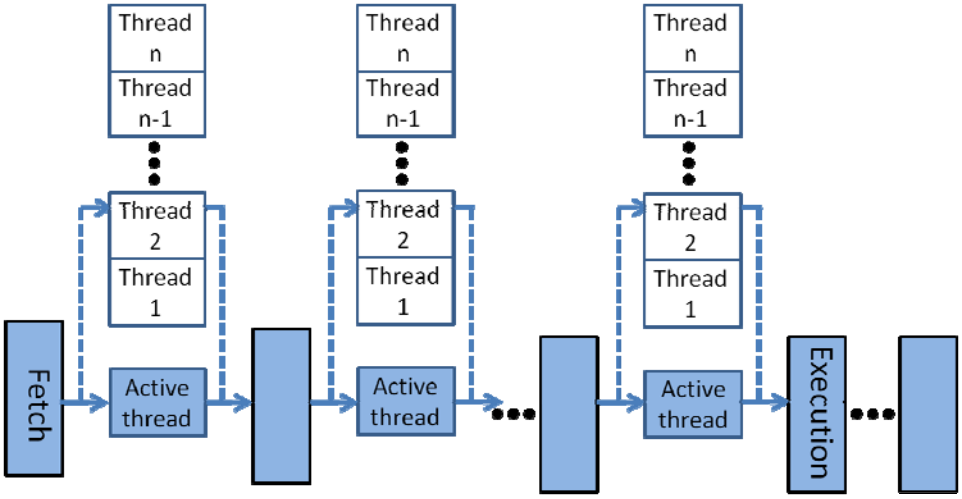


Figure 10. Continuous Flow Multithreading structure. A multistate pipeline register (MPR) is located between each two pipeline stages instead of a conventional pipeline register. The MPR stores the state of instructions from all supported threads within the machine when only a single thread is active at a time.

Chapter 3 Published Papers

This chapter contains the full collection of the scientific papers that were published during the thesis in scientific journals and refereed conference proceedings. Each subchapter is devoted to a different aspect of our research.

3.1 Device Characteristic and Modeling

This section contains the following papers:

- S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.
- S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Models of Memristors for SPICE Simulations," *Proceedings of the IEEE Convention of Electrical and Electronics Engineers in Israel*, pp. 1-5, November 2012.
- S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The Desired Memristor for Circuit Designers," *IEEE Circuits and Systems Magazine*, Vol. 13, No. 2, pp. 17-22, second quarter 2013.

TEAM: ThrEshold Adaptive Memristor Model

Shahar Kvatinsky, Eby G. Friedman, *Fellow, IEEE*, Avinoam Kolodny, *Senior Member, IEEE*, and Uri C. Weiser, *Fellow, IEEE*

***Abstract*—Memristive devices are novel devices, which can be used in applications ranging from memory and logic to neuromorphic systems. A memristive device offers several advantages: nonvolatility, good scalability, effectively no leakage current, and compatibility with CMOS technology, both electrically and in terms of manufacturing. Several models for memristive devices have been developed and are discussed in this paper. Digital applications such as memory and logic require a model that is highly nonlinear, simple for calculations, and sufficiently accurate. In this paper, a new memristive device model is presented—TEAM, ThrEshold Adaptive Memristor model. This model is flexible and can be fit to any practical memristive device. Previously published models are compared in this paper to the proposed TEAM model. It is shown that the proposed model is reasonably accurate and computationally efficient, and is more appropriate for circuit simulation than previously published models.**

***Index Terms*—Memristive systems, memristor, SPICE, window function.**

I. Introduction

MEMRISTORS are passive two-port elements with variable resistance (also known as a memristance) [1]. Changes in the memristance depend upon the history of the device (e.g., the memristance may depend on the total charge passed through the device, or alternatively, on the integral over time of the applied voltage between the ports of the device).

Formally, a current-controlled time-invariant memristive system [2] is represented by

$$\frac{dw}{dt} = f(w, i), \tag{1}$$

$$v(t) = R(w, i) \cdot i(t), \tag{2}$$

where w is an internal state variable, $i(t)$ is the memristive device current, $v(t)$ is the memristive device voltage, $R(w, i)$ is the memristance, and t is time. The terms memristor and memristive systems are often used interchangeably to describe memristive systems [2]. While there are discussions in the literature about specific definitions [29], [30], in this paper we use the term "memristive device" to describe all devices within these categories.

Since Hewlett-Packard announced the fabrication of a working memristive device in 2008 [3], there has been an increasing interest in memristors and memristive systems. New devices exhibiting memristive behavior have been announced [4], [5], and existing devices such as spin-transfer torque magnetoresistive random access memory (STT-MRAM) have been redescribed in terms of memristive systems [6].

Memristive devices can be used for a variety of applications such as memory [7], neuromorphic systems [8], analog circuits (e.g., see [9]), and logic design [10], [27]. Different characteristics are important for the effective use of memristive devices in each of these applications, and an appropriate designer friendly physical model of a memristive device is therefore required.

In this paper, the characteristics of memristive devices are described in Section II. Previously published memristive device models are reviewed in Section III. TEAM—a new model that is preferable in terms of the aforementioned characteristics—is proposed in Section IV. In Section V, a comparison between these models is presented. The paper is summarized in Section VI.

II. Requirements for Memristive Device Characteristics

Different applications require different characteristics from the building blocks. Logic and memory applications, for example, require elements for computation and control, as well as the ability to store data after computation. These elements require sufficiently fast read and write times. The read mechanism needs to be nondestructive, i.e., the reading mechanism should not change the stored data while reading. To store a known digital state and maintain low sensitivity to variations in parameters and operating conditions, it is crucial that the stored data be distinct, i.e., the difference between different data must be sufficiently large. The transient power consumption while reading and writing, as well as static power consumption, are also critical issues.

Although the definition of a memristive system is quite broad, all memristive systems exhibit a variable resistance, which is related to an internal state variable. Memristive devices employed in practice exhibit a nonvolatile behavior. To provide a nondestructive read mechanism, the internal state variable needs to exhibit a nonlinear dependence on charge, i.e., changes in the state variable due to high currents should be significant, while changes due to low currents should be negligible. Other mechanisms where the state variables return to the original position after completing the read process may also require the nondestructive read mechanism. For certain applications such as analog counters, however, a linear dependence on charge is preferable, since the current is integrated during the counting process.

To store distinct Boolean data in a memristive device, a high ratio between the resistances (typically named R_{ON} and R_{OFF}) is necessary. Several additional characteristics are important for

Manuscript received January 17, 2012; revised April 08, 2012; accepted April 22, 2012. Date of publication November 15, 2012; date of current version January 04, 2013. This work was supported in part by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, and by Intel Grant 864-737-13. This paper was recommended by Associate Editor F. Lustenberger.

S. Kvatinsky, A. Kolodny, and U. C. Weiser are with the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: skva@tx.technion.ac.il).

E. G. Friedman is with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627, USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2012.2215714

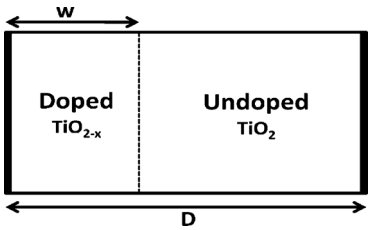


Fig. 1. Linear ion drift memristive device model. The device is composed of two regions: doped and undoped. The total resistance of the device is the sum of the resistance of both regions.

all applications, such as low power consumption, good scalability, and compatibility with conventional CMOS.

These characteristics exist in memristive devices. STT-MRAM exhibits these characteristics except for the high off/on resistance ratio [11]. To design and analyze memristive device-based circuits and applications, a model exhibiting these traits is required.

III. Previously Proposed Memristive Device Models

A. Requirements From an Effective Memristive Device Model

An effective memristive device model needs to satisfy several requirements: it must be sufficiently accurate and computationally efficient. It is desirable for the model to be simple, intuitive, and closed-form. It is also preferable for the model to be general so that it can be tuned to suit different types of memristive devices.

B. Linear Ion Drift Model

A linear ion drift model for a memristive device is suggested in [3]. In this model, one assumption is that a device of physical width D contains two regions, as shown in Fig. 1. One region of width w (which acts as the state variable of the system) has a high concentration of dopants (originally oxygen vacancies of TiO_2, namely TiO_{2-x}). The second region of width $D-w$ is an oxide region (originally TiO_2). The region with the dopants has a higher conductance than the oxide region, and the device is modeled as two resistors connected in series. Several assumptions are made: ohmic conductance, linear ion drift in a uniform field, and the ions have equal average ion mobility μ_V. Equations (1) and (2) become, respectively,

$$\frac{dw}{dt} = \mu_v \frac{R_{ON}}{D} i(t), \tag{3}$$

$$v(t) = \left(R_{ON} \frac{w(t)}{D} + R_{OFF} \left(1 - \frac{w(t)}{D} \right) \right) \cdot i(t), \tag{4}$$

where R_{ON} is the resistance when $w(t) = D$, and R_{OFF} is the resistance when $w(t) = 0$. The state variable $w(t)$ is limited to the physical dimensions of the device, i.e., the value is within the interval $[0, D]$. To prevent w from growing beyond the physical device size, the derivative of w is multiplied by a window function, as discussed in Section III-C. The I-V curve of a linear ion drift memristive device for sinusoidal and rectangular waveform inputs is shown in Fig. 2.

C. Window Function

In the linear ion drift model, the permissible value of the state variable is limited to the interval $[0, D]$. To satisfy these bounds, (3) is multiplied by a function that nullifies the derivative, and

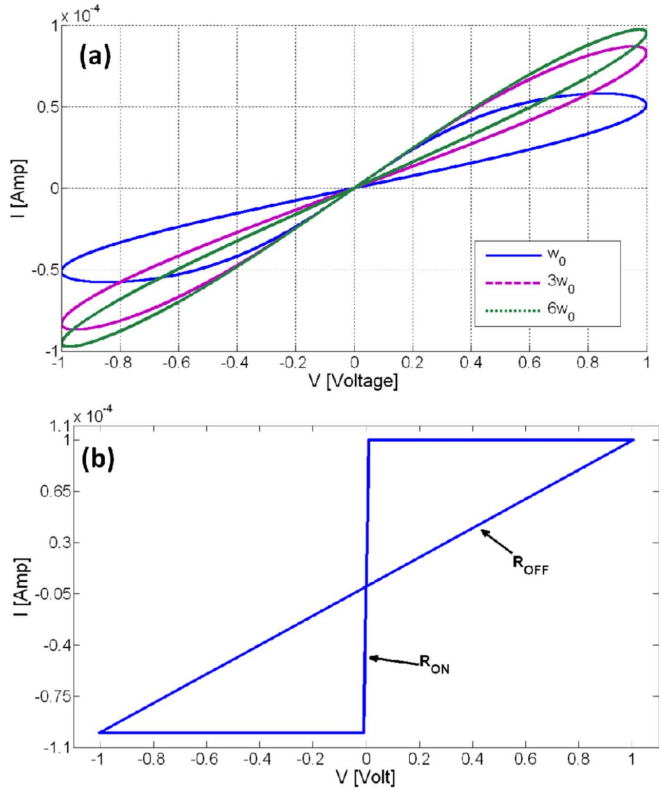


Fig. 2. Linear ion drift model I-V curve. $D = 10$ nm, $R_{ON} = 100\ \Omega$, $R_{OFF} = 16\ \mathrm{k}\Omega$, $\mu_V = 10^{-14}\ \mathrm{m^2 s^{-1} V^{-1}}$, and $w_0 = 1\ \mathrm{Rad/s}$. (a) Sinusoidal voltage input for several frequencies ω_0, $3\omega_0$, and $6\omega_0$, and (b) rectangular waveform current input.

forces (3) to be identical to zero when w is at a bound. One possible approach is an ideal rectangular window function (the function where the value is 1 for any value of the state variable, except at the boundaries where the value is 0). It is also possible to add a nonlinear ion drift phenomenon, such as a decrease in the ion drift speed close to the bounds, with a different window [12],

$$f(w) = 1 - \left(\frac{2w}{D} - 1 \right)^{2p}, \tag{5}$$

where p is a positive integer. For large values of p, the window function becomes similar to a rectangular window function, and the nonlinear ion drift phenomenon decreases, as shown in Fig. 3.

The window function in (5) exhibits a significant problem for modeling practical devices, since the derivative of w is forced to zero and the internal state of the device cannot change if w reaches one of the bounds. To prevent this modeling inaccuracy, a different window function has been proposed [13],

$$f(w) = 1 - \left(\frac{w}{D} - stp(-i) \right)^{2p}, \tag{6}$$

$$stp(i) = \begin{cases} 1, & i \geq 0 \\ 0, & i < 0, \end{cases} \tag{7a, 7b}$$

where i is the memristive device current. This function is shown in Fig. 4. In the original definition, these window functions do not have a scale factor and therefore cannot be adjusted, i.e., the maximum value of the window function cannot be changed to a value lower or greater than one. To overcome this limitation, a minor enhancement—adding a multiplicative scale factor to

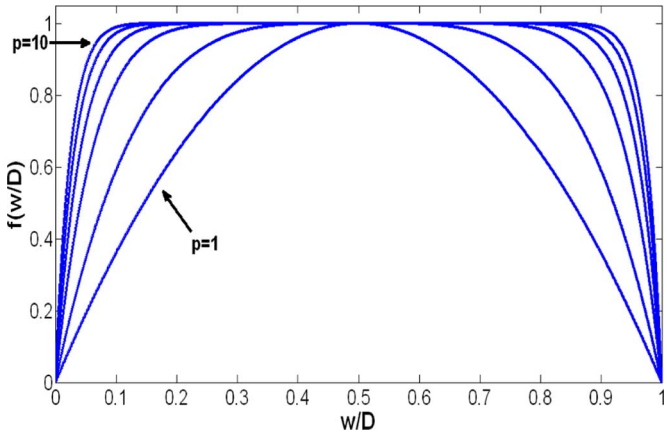


Fig. 3. Window function described by (5) according to [12] for several values of p.

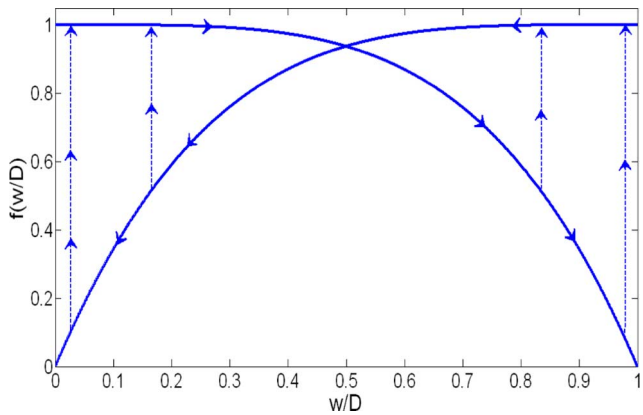


Fig. 4. Window function described by (6) according to [13].

the window function, has recently been proposed [14]. The proposed window function in [14] is

$$f(w) = j\left(1 - \left[\left(\frac{w}{D} - 0.5\right)^2 + 0.75\right]^p\right), \qquad (8)$$

where j is a control parameter which determines the maximum value of $f(w)$ (in this function, the maximum value can be smaller or larger than one). This function is shown in Fig. 5.

While these window functions alleviate the bounds issue and suggest a nonlinear phenomenon, these functions do not exhibit full nonlinear ion drift behavior since the model ignores the nonlinear dependence of the state derivative on the current. A linear ion drift model with a window function does not therefore fully model nonlinear ion drift behavior.

D. Nonlinear Ion Drift Model

While the linear ion drift model is intuitive and satisfies the basic memristive system equations, experiments have shown that the behavior of fabricated memristive devices deviates significantly from this model and is highly nonlinear [15], [16]. The nonlinear I-V characteristic is desirable for logic circuits, and hence more appropriate memristive device models have been proposed. In [17], a model is proposed based on the experi-

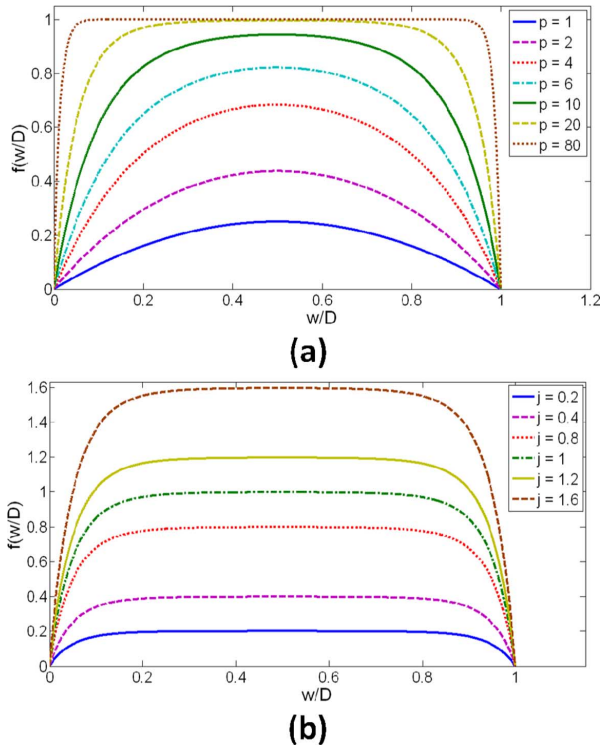


Fig. 5. Window function described by (8) according to [14]. (a) Varying p, and (b) varying j.

mental results described in [15]. The relationship between the current and voltage is

$$i(t) = w(t)^n \beta \sinh\left(\alpha v(t)\right) + \chi\left[\exp\left(\gamma v(t)\right) - 1\right], \qquad (9)$$

where α, β, γ and χ are experimental fitting parameters, and n is a parameter that determines the influence of the state variable on the current. In this model, the state variable w is a normalized parameter within the interval $[0, 1]$. This model assumes an asymmetric switching behavior. When the device is in the ON state, the state variable w is close to one and the current is dominated by the first expression in (9), $\beta \sinh(\alpha v(t))$, which describes a tunneling phenomenon. When the device is in the OFF state, the state variable w is close to zero and the current is dominated by the second expression in (9), $\chi[\exp(\lambda v(t)) - 1]$, which resembles an ideal diode equation.

This model assumes a nonlinear dependence on voltage in the state variable differential equation,

$$\frac{dw}{dt} = a \cdot f(w) \cdot v(t)^m, \qquad (10)$$

where a and m are constants, m is an odd integer, and $f(w)$ is a window function. The I-V relationship of a nonlinear ion drift memristive device for sinusoidal and rectangular waveform inputs is illustrated in Fig. 6. A similar model is proposed by the same authors in [28]. In this model, the same I-V relationship is described with a more complex state drift derivative.

E. Simmons Tunnel Barrier Model

Linear and nonlinear ion drift models are based on representing the two regions of oxide and doped oxide as two resis-

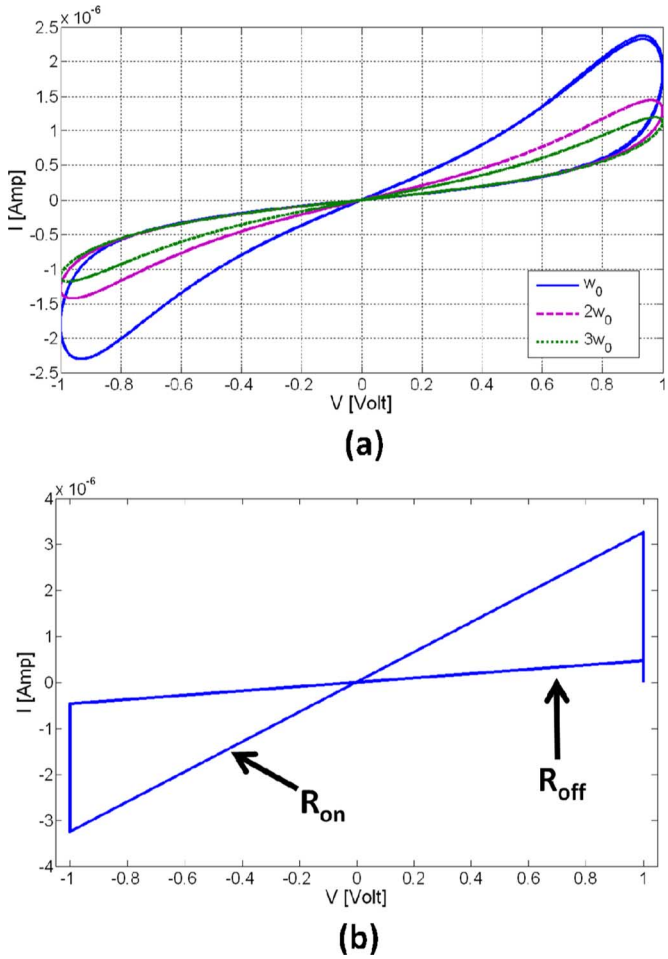


Fig. 6. Nonlinear ion drift model I-V curve. $m = 5, n = 2, a = 1\ \mathrm{V}^{-m}\mathrm{s}^{-1}$, $\beta = 0.9\ \mu\mathrm{A}, \gamma = 4\ \mathrm{V}^{-1}, \chi = 10^{-4}\ \mu\mathrm{A}$, and $\alpha = 2\ \mathrm{V}^{-1}$. (a) Sinusoidal voltage input for several frequencies ω_0, $2\omega_0$, and $3\omega_0$, and (b) rectangular waveform of input voltage.

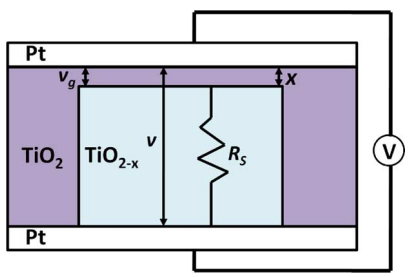


Fig. 7. Physical model of Simmons tunnel barrier memristive device. The state variable x is the width of the oxide region, V is the applied voltage on the device, v_g is the voltage in the undoped region, and v is the internal voltage in the device.

tors in series. A more accurate physical model was proposed in [18]. This model assumes nonlinear and asymmetric switching behavior due to an exponential dependence of the movement of the ionized dopants, namely, changes in the state variable. In this model, rather than two resistors in series as in the linear drift model, there is a resistor in series with an electron tunnel barrier, as shown in Fig. 7. The state variable x is the Simmons tunnel barrier width [19] (note that a different notation for the state variable is used to prevent confusion with the role of the state variable in the linear ion drift model). In this case, the derivative

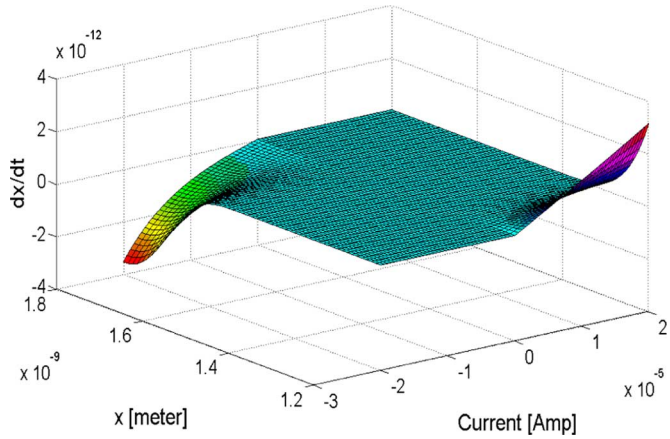


Fig. 8. Derivative of the state variable x as described in (11). The fitting parameters are $c_{off} = 3.5\ \mu\mathrm{m/s}$, $i_{off} = 115\ \mu\mathrm{A}$, $a_{off} = 1.2\ \mathrm{nm}$, $c_{on} = 40\ \mu\mathrm{m/s}$, $i_{on} = 8.9\ \mu\mathrm{A}$, $a_{on} = 1.8\ \mathrm{nm}$, $b = 500\ \mu\mathrm{A}$, and $w_c = 107\ \mathrm{pm}$.

of x can be interpreted as the oxygen vacancy drift velocity, and is

$$\frac{dx(t)}{dt} = \begin{cases} c_{off} \sinh\left(\dfrac{i}{i_{off}}\right) \exp\left[-\exp\left(\dfrac{x - a_{off}}{w_c} - \dfrac{|i|}{b}\right) - \dfrac{x}{w_c}\right], & i > 0 \quad (11a) \\ c_{on} \sinh\left(\dfrac{i}{i_{on}}\right) \exp\left[-\exp\left(-\dfrac{x - a_{on}}{w_c} - \dfrac{|i|}{b}\right) - \dfrac{x}{w_c}\right], & i < 0, \quad (11b) \end{cases}$$

where c_{off}, c_{on}, i_{off}, i_{on}, a_{off}, a_{on}, w_c, and b are fitting parameters. Equation (11) is illustrated in Fig. 8 for the measured fitting parameters reported in [18]. The physical phenomena behind the behavior shown in (11) are not yet fully understood, but considered to be a mixture of nonlinear drift at high electric fields and local Joule heating enhancing the oxygen vacancies. In practical memristive devices, the ON switching is significantly faster than the OFF switching because of the diffusion of the oxygen vacancies from TiO_{2-x} to TiO_2, and the drift of the oxygen vacancies due to the internal electric field is different for positive and negative voltages. For a negative voltage (lower x), the drift of the oxygen vacancies and the diffusion are in the same direction, while for a positive voltage, the direction of diffusion and drift is opposite [20]. The parameters c_{off} and c_{on} influence the magnitude of the change of x. The parameter c_{on} is an order of magnitude larger than the parameter c_{off}. The parameters i_{off} and i_{on} effectively constrain the current threshold. Below these currents, the change in the derivative of x is neglected. A current threshold phenomenon is desirable for digital applications. The parameters a_{off} and a_{on} force, respectively, the upper and lower bounds for x. Because of the exponential dependence on $x - a_{off}$ or $x - a_{on}$, the derivative of the state variable is significantly smaller for the state variable within the permitted range. There is therefore no need for a window function in this model.

In this model, the relationship between the current and voltage is shown as an implicit equation based on the Simmons tunneling model [19],

$$\begin{aligned} i(t) = {} & \tilde{A}(x, v_g)\phi_1(v_g, x) \exp\left(-B(v_g, x) \cdot \phi_1(v_g, x)^{1/2}\right) \\ & - \tilde{A}(x, v_g)\left(\phi_1(v_g, x) + e|v_g|\right) \\ & \times \exp\left(-B(v_g, x) \cdot \left(\phi_1(v_g, x) + ev_g\right)^{1/2}\right), \quad (12) \\ v_g = {} & v - i(t)R_s, \quad (13) \end{aligned}$$

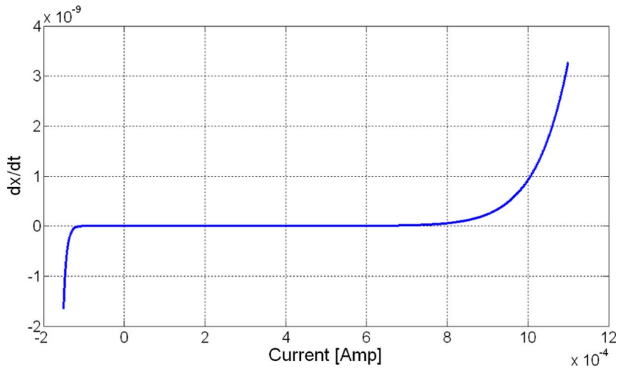


Fig. 9. Derivative of the state variable x as described in (11) under the assumption of a small change in $x (x \sim 1.5\ \mathrm{nm})$. Note that the device exhibits a threshold current. The same fitting parameters as used in Fig. 8 are used.

where v is the internal voltage on the device, which is not necessarily equal to the applied voltage on the device V (i.e., the external voltage V and the internal voltage v are not necessarily the same [18]).

IV. Threshold Adaptive Memristor (TEAM) Model

In this section, TEAM, a novel memristive device model, is presented. The integral portion of the TEAM model is based on an expression for the derivative of the internal state variable that can be fitted to any memristive device type. Unlike other memristive device models, the current-voltage relationship is undefined and can be freely chosen from any current-voltage relationship. Several examples of possible current-voltage relationships are described in Section IV-B. This relationship is not limited to these examples. In Section IV-A, the disadvantages of the aforementioned models and the need for such a model are explained. The derivative of the internal state variable of the memristive device [the relevant expression for (1)] and examples of the current-voltage relationship [the relevant expression for (2)] are described, respectively, in Section IV-B and IV-C. Proper fitting of the Simmons tunnel barrier model to the TEAM model is presented in Section IV-D, as well as the proper window function for this fitting.

A. Need for a Simplified Model

The Simmons tunnel barrier model is, to the authors' best knowledge, the most accurate physical model of a TiO_2 memristive device. This model is however quite complicated, without an explicit relationship between current and voltage, and not general in nature (i.e., the model fits only a specific type of memristive device). A complex SPICE model of the Simmons tunnel barrier model is presented in [21]. This model is also computational inefficient. A model with simpler expressions rather than the complex equations in the Simmons tunnel barrier model is therefore desired. Yet the accuracy of the simple model must be adequate. This simplified model represents the same physical behavior, but with simpler mathematical functions. In Section V, simplifying assumptions are introduced. Namely, no change in the state variable is assumed below a certain threshold, and a polynomial dependence rather than an exponential dependence is used. These assumptions are applied to support simple analysis and computational efficiency.

B. State Variable Derivative in TEAM Model

Note in Fig. 9 and (11) that because of the high nonlinear dependence of the memristive device current, the memristive device can be modeled as a device with threshold currents. This approximation is similar to the threshold voltage approximation in MOS transistors. This approximation is justified, since for small changes in the electric tunnel width, separation of variables can be performed. The dependence of the internal state derivative on current and the state variable itself can be modeled as independently multiplying two independent functions; one function depends on the state variable x and the other function depends on the current.

Under these assumptions, the derivative of the state variable for the simplified proposed model is

$$\frac{dx(t)}{dt} = \begin{cases} k_{off} \cdot \left(\dfrac{i(t)}{i_{off}} - 1 \right)^{\alpha_{off}} \cdot f_{off}(x), & 0 < i_{off} < i \quad \text{(14a)} \\ 0, & i_{on} < i < i_{off} \quad \text{(14b)} \\ k_{on} \cdot \left(\dfrac{i(t)}{i_{on}} - 1 \right)^{\alpha_{on}} \cdot f_{on}(x), & i < i_{on} < 0, \quad \text{(14c)} \end{cases}$$

where k_{off}, k_{on}, α_{off}, and α_{on} are constants, i_{off} and i_{on} are current thresholds, and x is the internal state variable, which represents the effective electric tunnel width. The constant parameter k_{off} is a positive number, while the constant parameter k_{on} is a negative number. The functions $f_{off}(x)$ and $f_{on}(x)$ represent the dependence on the state variable x. These functions behave as the window functions described in Section II, which constrain the state variable to bounds of $x \in [x_{on}, x_{off}]$. Alternatively, these functions can be different functions of x. The functions $f_{on}(x)$ and $f_{off}(x)$ are not necessarily equal, since the dependence on x may be asymmetric (as in the Simmons tunnel barrier model). Note that the role of x in this model is opposite to w in the linear ion drift model.

C. Current—Voltage Relationship in TEAM Model

Assume the relationship between the voltage and current of a memristive device is similar to (4). The memristance changes linearly in x, and (2) becomes

$$v(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{x_{off} - x_{on}} (x - x_{on}) \right] \cdot i(t). \quad (15)$$

The reported change in the resistance however is an exponential dependence on the state variable [18], since the memristance, in practical memristive devices, is dependent on a tunneling effect, which is highly nonlinear. If (12) describes the current-voltage relationship in the model, the model becomes inefficient in terms of computational time and is also not general. Therefore, any change in the tunnel barrier width changes the memristance, and is assumed to change in an exponential manner. Under this assumption, (2) becomes

$$v(t) = R_{ON} e^{(\lambda / x_{off} - x_{on})(x - x_{on})} \cdot i(t), \quad (16)$$

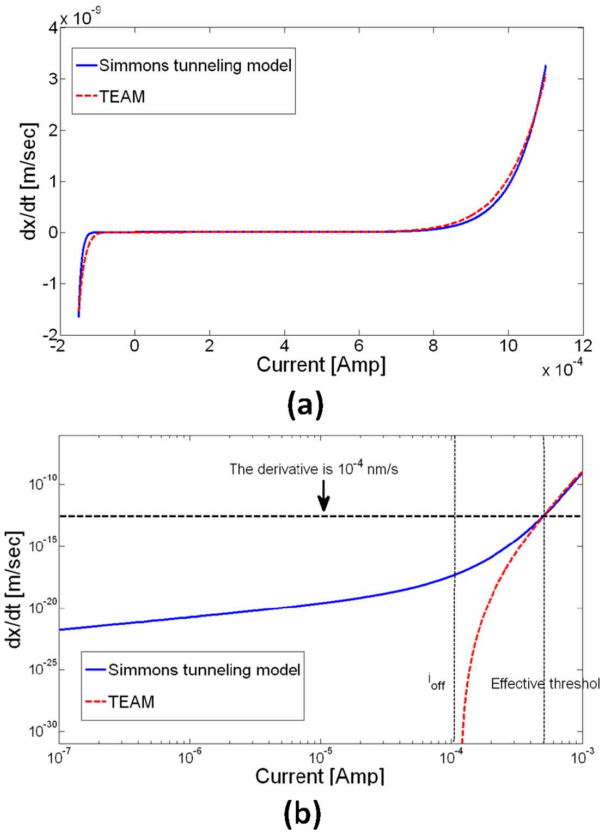


Fig. 10. Fitting between the derivative of the state variable x in the Simmons tunnel barrier memristive device model and the TEAM model. The same fitting parameters as used in Fig. 8 are used for the Simmons tunnel barrier model. (a) The fitting parameters for the proposed model are $k_{off} = 1.46e\text{-}9$ nm/s, $\alpha_{off} = 10, i_{off} = 115\,\mu$A, $k_{on} = -4.68e\text{-}13$ nm/s, $\alpha_{on} = 10$, and $i_{on} = 8.9\,\mu$A. (b) Fitting procedure in a logarithmic scale. The operating current range is assumed to be 0.1 μA to 1 mA and the neglected value for the derivative of the state variable is assumed to be 10^{-4} nm/s. For any desired current range, the proper fitting parameters can be evaluated to maintain an accurate match between the models. For the aforementioned parameters, a reasonable current threshold is 0.5 mA (marked as the effective threshold in the figure).

where λ is a fitting parameter, and R_{ON} and R_{OFF} are the equivalent effective resistance at the bounds, similar to the notation in the linear ion drift model, and satisfy

$$\frac{R_{OFF}}{R_{ON}} = e^{\lambda}. \tag{17}$$

D. Fitting the Simmons Tunnel Barrier Model to the TEAM Model

The TEAM model is inspired by the Simmons tunnel barrier model. However, to use this model for practical memristive devices, similar to the Simmons tunnel barrier model, a fit to the TEAM model needs to be accomplished. Since (14) is derived from a Taylor series, for any desired range of memristive device current, $\lambda, k_{off}, k_{on}, \alpha_{off}$, and α_{on} can be evaluated to achieve a sufficiently accurate match between the models. As the desired operating current range for the memristive device is wider, to maintain sufficiently accuracy, the required α_{off} and α_{on} are higher, thereby increasing the computational time. The proper fitting procedure to the current threshold is to plot the derivative of the exact state variable in the actual operating range of the current, and decide what value of the state variable derivative is effectively zero (i.e., the derivative of the state variable is

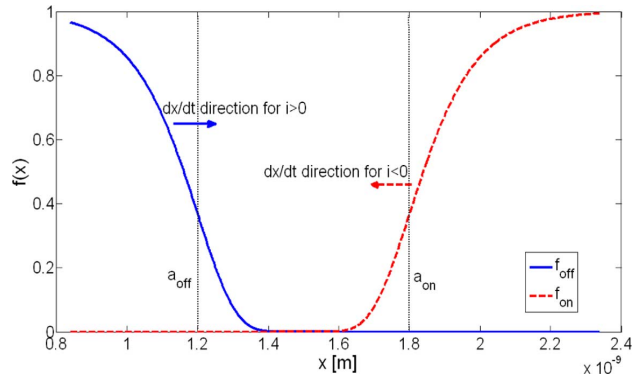


Fig. 11. Proposed $f_{on}(x)$ and $f_{off}(x)$ based on (18) and (19). These functions represent the dependence on x in (14) and also force bounds for x since $f_{off}(x)$ is used when dx/dt is positive and is zero around a_{on}, and vice versa for $f_{on}(x)$.

significantly smaller and can therefore be neglected). The current at this effective point is a reasonable value of the current threshold. In this paper, the parameters i_{on} and i_{off} are chosen as these current thresholds, since these terms represent the exponential dependence of the derivative on the state variable of the current in the Simmons tunnel barrier model. A fit of the Simmons tunnel barrier model to the TEAM model is shown in Fig. 10(a). The proper current threshold fitting procedure is shown in Fig. 10(b). Note that a reasonable current threshold can be higher than i_{off}.

As mentioned in Section IV-B, the functions $f_{off}(x)$ and $f_{on}(x)$ are window functions, or alternatively, functions that fit the Simmons tunnel barrier model based upon the separation of variables of (11). These functions represent the dependence of the derivative in the state variable x. Based on the fitting parameters reported in [18], possible functions $f_{off}(x)$ and $f_{on}(x)$ are, respectively,

$$f_{off}(x) = \exp\left[-\exp\left(\frac{x - a_{off}}{w_c}\right)\right], \tag{18}$$

$$f_{on}(x) = \exp\left[-\exp\left(-\frac{x - a_{on}}{w_c}\right)\right]. \tag{19}$$

The determination process for (18) and (19) is presented in Appendix A. Note that (18) and (19) maintain the limitation of certain bounds for the state variable x since the derivative of x around a_{on} when using (18) and (19) is effectively zero for positive current (f_{off} is practically zero) and negative for negative current. x can only be reduced. The value of x can be increased for values of x around a_{off}. Therefore, a reasonable value for the state variable bounds x_{on} and x_{off} is, respectively, a_{off} and a_{on}. Although the proposed function limits the bounds of the state variable, there is no problem when the bounds are exceeded, unlike other window functions. This characteristic is useful for simulations, where the bounds can be exceeded due to the discrete nature of simulation engines. The proposed terms, f_{off} and f_{on}, are illustrated in Fig. 11.

The I-V relationship and state variable behavior of the proposed model are shown in Figs. 12 and 13 for an ideal rectangular window function and the proposed window function. Note in Figs. 12 and 13 that there is a performance difference between the different window functions. Due to the significant nonlinearity, the proposed window function constrains the state

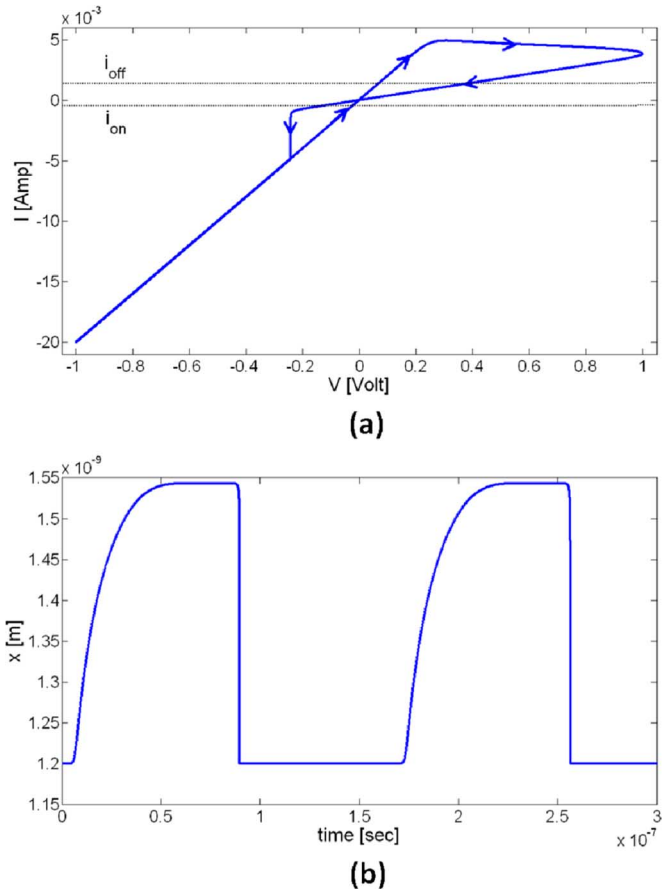


Fig. 12. The TEAM model driven with a sinusoidal input of 1 volt using the same fitting parameters as used in Fig. 10, $R_{ON} = 50\ \Omega$, $R_{OFF} = 1\ \mathrm{k}\Omega$, and an ideal rectangular window function for $f_{on}(x)$ in (19) and $f_{off}(x)$ in (18). (a) I-V curve, and (b) state variable x. Note that the device is asymmetric, i.e., switching OFF is slower than switching ON.

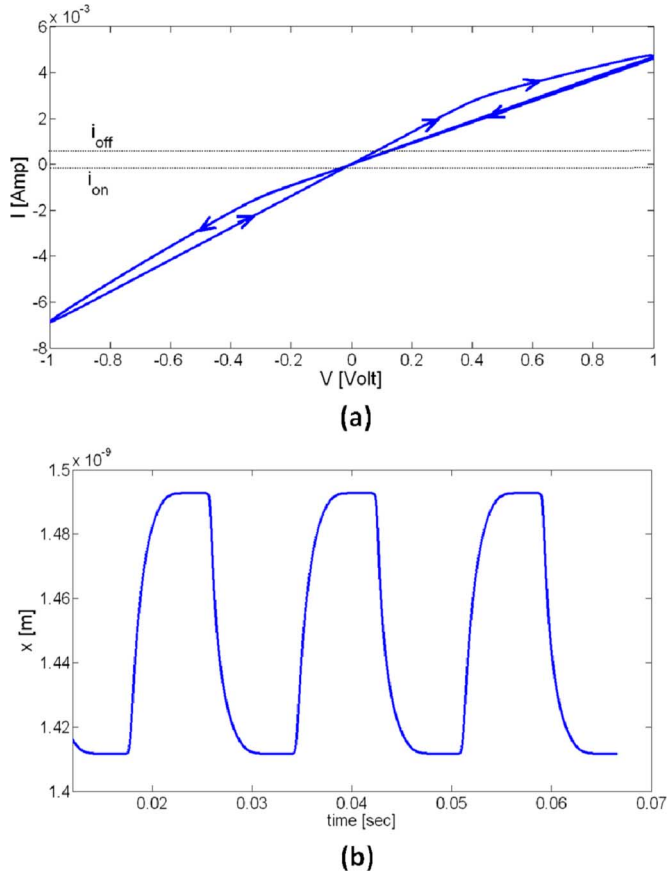
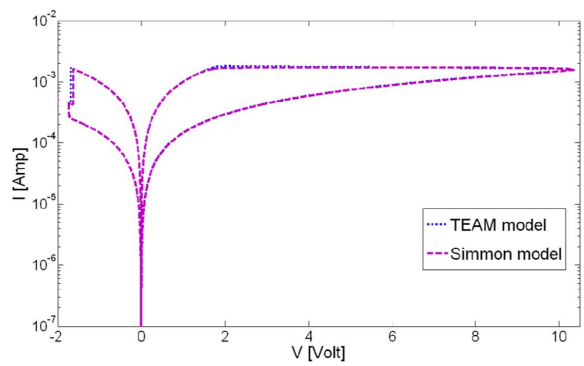


Fig. 13. The TEAM model driven with a sinusoidal input of 1 volt using the same fitting parameters as used in Fig. 10, $R_{ON} = 50\ \Omega$, $R_{OFF} = 1\ \mathrm{k}\Omega$, proposed $f_{on}(x)$ in (19), and $f_{off}(x)$ in (18) with the same parameters used in Fig. 8. (a) I-V curve, and (b) state variable x. Note that the device is asymmetric, i.e., switching OFF is slower than switching ON.

variable to a small range, and the memristive devices are activated within a significantly smaller time scale as compared to an ideal rectangular window function. The required conditions for a sufficient fit of the TEAM model to the Simmons tunnel barrier model, as described in Appendix A, cannot be maintained for a symmetric input voltage due to the asymmetry of the Simmons tunnel model. The required conditions for a sufficient fit are therefore not maintained in Fig. 13. These conditions are however maintained in Fig. 14, where the behavior of the TEAM model and the Simmons tunnel barrier model is compared and exhibits excellent agreement. While the proposed model fits the Simmons Tunnel Barrier model, the TEAM model is general and flexible. The model can fit different physical memristive device models, including other types of memristive devices, such as STT-MRAM and Spintronic memristors [6], [24].

V. Comparison Between the Models

A comparison between the different memristive device models is listed in Table I and a comparison between different window functions is listed in Table II. A comparison of the accuracy and complexity between the Simmons tunnel barrier memristive device and TEAM models is shown in Fig. 14. The TEAM model can improve the simulation runtime by 47.5% and is sufficiently accurate, with a mean error of 0.2%. These results are dependent on the particular TEAM parameters. A lower value for α_{ON} and α_{OFF} produces lower accuracy and



Maximum State Variable Difference [%]	12.77%
Mean State Variable Difference [%]	0.2%
Maximum Run Time Improvement [%]	47.5%

(b)

Fig. 14. TEAM model fitted to Simmons tunnel barrier model. (a) I-V curve for both models, and (b) fitting accuracy in terms of internal state variable x and maximum improvement in runtime for MATLAB simulations. The state variable average and maximum differences are, respectively, 0.2% and 12.77%. The TEAM fitting parameters are $R_{ON} = 1\ \mathrm{k}\Omega$, $R_{OFF} = 100\ \mathrm{k}\Omega$, $k_{on} = 4.13e\text{-}33\ \mathrm{nm/s}$, $\alpha_{on} = 25$, $k_{off} = 4.13e\text{-}33\ \mathrm{nm/s}$, $\alpha_{off} = 25$, $i_{off} = 115\ \mu\mathrm{A}$, and $i_{on} = 8.9\ \mu\mathrm{A}$.

enhanced computational runtime. The TEAM model satisfies the primary equations of a memristive system as described in

TABLE I
COMPARISON OF DIFFERENT MEMRISTIVE DEVICE MODELS

Model	Linear ion drift [3]	Nonlinear ion drift [17]	Simmons tunneling barrier [18]	Yakopcic *et al* [25]	TEAM
State variable	$0 \le w \le D$ Doped region physical width	$0 \le w \le 1$ Doped region normalized width	$a_{off} \le x \le a_{on}$ Undoped region width	$0 \le x \le 1$ No physical explanation	$x_{on} \le x \le x_{off}$ Undoped region width
Control mechanism	Current controlled	Voltage controlled	Current controlled	Voltage controlled	Current controlled
Current-voltage relationship and memristance deduction	Explicit	I-V relationship – explicit Memristance deduction - ambiguous	Ambiguous	Ambiguous	Explicit
Matching memristive system definition	Yes	No	No	No	Yes
Generic	No	No	No	Moderate	Yes
Accuracy comparing practical memristive devices	Lowest accuracy	Low accuracy	Highest accuracy	Moderate accuracy	Sufficient accuracy
Threshold exists	No	No	Practically exists	Yes	Yes

TABLE II
COMPARISON OF DIFFERENT WINDOW FUNCTIONS

Function	Joglekar [12]	Biolek [13]	Prodromakis [14]	TEAM
f(x)/f(w)	$f(w) = 1-(2w/D-1)^{2p}$	$f(w) = 1-(w/D-stp(-i))^{2p}$	$f(w)=j(1-[(w-0.5)^2+0.75]^p)$	$f_{on,off}=exp[-exp(\|x-x_{on,off}\|/w_c)]$
Symmetric	Yes	Yes	Yes	Not necessarily
Resolve boundary conditions	No	Yes	Practically yes	Practically yes
Impose nonlinear drift	Partially	Partially	Partially	Yes
Scale factor $f_{max} < 1$	No	No	Yes	No
Fits memristive device model	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	TEAM for Simmons tunneling barrier fitting

(1) and (2), and the convergence conditions and computational efficiency required by simulation engines.

The TEAM model accurately characterizes not only the Simmons tunnel barrier model, but also a variety of different models. For example, the TEAM model can be fitted to the linear ion drift behavior, where

$$k_{on} = k_{off} = \mu_v \frac{R_{ON}}{D} i_{on}, \tag{20}$$

$$\alpha_{on} = \alpha_{off} = 1, \tag{21}$$

$$i_{on} = i_{off} \to 0, \tag{22}$$

$$x_{on} = D, \tag{23}$$

$$x_{off} = 0, \tag{24}$$

$$x = D - w. \tag{25}$$

To include memristive devices into the circuit design process, these models need to be integrated into a CAD environment, such as SPICE. There are several proposed SPICE macromodels for the linear ion drift model [13], [22] and the nonlinear ion drift model [17]. A SPICE model for the Simmons tunneling barrier model has recently been proposed [21], but is complicated and inefficient in terms of computational time. Another simplified model has recently been proposed, assuming voltage threshold and an implicit memristance [25]. In this model, the current and voltage are related through a hyperbolic sine and the derivative of the state variable is an exponent. This model is less general than the TEAM model and more complex in terms of computational time (the model uses sinh and exponents rather than polynomials as in the TEAM model). The model is also less accurate than the TEAM model when fitting the model to the Simmons tunnel barrier model.

The TEAM model can be described in a SPICE macromodel similar to the proposed macromodel in [23], as shown in Fig. 15. In this macromodel, the internal state variable is represented by the voltage across the capacitor C and the bounds of the state variable are enforced by diodes $D1$ and $D2$. A Verilog-A model is however chosen because it is more efficient in terms of computational time than a SPICE macromodel, while providing similar accuracy. A Verilog-A form of the model, as described in this paper, has been implemented. The code for these models can be found in [26]. Although the state variable derivative in the TEAM model is not a smooth function, it is a continuous function based only on polynomial functions. The Verilog-A model has been tested in complex simulations (hundreds of memristive devices) and did not exhibit any convergence issues.

VI. CONCLUSIONS

Different memristive device models are described in this paper—linear ion drift, nonlinear ion drift, Simmons tunnel barrier, and TEAM (ThrEshold Adaptive Memristor), as well as different window functions. The TEAM model is a flexible and convenient model that can be used to characterize a variety of different practical memristive devices. This model suggests a memristive device should exhibit a current threshold and nonlinear dependence on the charge, as well as a dependence on the state variable.

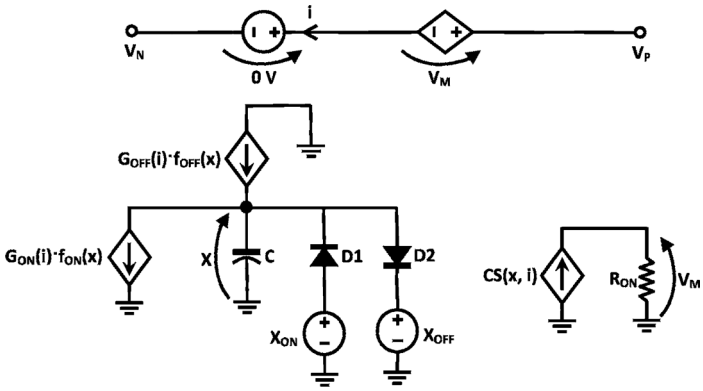


Fig. 15. TEAM SPICE macromodel. The state variable x is the voltage across the capacitor $C = 1$ F. The initial voltage is the initial state variable. $D1$ and $D2$ constrain the bounds of the state variable to the value of the voltage sources x_{ON} and x_{OFF}. $G_{ON}(i)$ and $G_{OFF}(i)$ are the relevant functions from (14). $CS(x, i)$ is determined from the current-voltage relationship, and is $i \cdot \exp[\lambda(x - x_{on})/(x_{off} - x_{on})]$ for the current-voltage relationship in (16). V_N and V_P are, respectively, the negative and positive ports of the memristive device, and i is the memristive device current.

A comparison between the TEAM model and other memristive device models is presented. The TEAM model is simple, flexible, and general. While the simplicity of this model improves the efficiency of the simulation process, the model is sufficiently accurate, exhibiting an average error of only 0.2% as compared to the Simmons tunnel barrier state variable. This model fits practical memristive devices better than previously proposed models. This model is suitable for memristive device-based circuit design and has been implemented in Verilog-A for SPICE simulations.

Appendix
Appropriate Fitting Window Function to the Simmons Tunnel Barrier Model

The purpose of this appendix is to determine a proper window function $f(x)$ that provides a sufficient fit to the Simmons tunnel barrier model. To determine a reasonable approximation, parameter values from [18] are used. From (11a) and (11b), the derivative of the state variable x is

$$\frac{dx(t)}{dt} = \begin{cases} C_{off}\sinh\left(\dfrac{i}{i_{off}}\right)\exp\left[-\exp\left(\dfrac{x-a_{off}}{w_c}-\dfrac{|i|}{b}\right)-\dfrac{x}{w_c}\right], & i>0 \quad \text{(A.1.a)} \\ C_{on}\sinh\left(\dfrac{i}{i_{on}}\right)\exp\left[-\exp\left(-\dfrac{x-a_{on}}{w_c}-\dfrac{|i|}{b}\right)-\dfrac{x}{w_c}\right], & i<0. \quad \text{(A.1.b)} \end{cases}$$

The derivative of the state variable is a multiplicand of two functions—a hyperbolic sine function which depends only on the current and an exponential function which depends on both the current and the state variable. To simplify (A.1) and to apply separation of variables, approximations

$$\frac{x - a_{off}}{w_c} \gg \frac{|i|}{b} \quad \text{(A.2.a)}$$

$$\frac{a_{on} - x}{w_c} \gg \frac{|i|}{b} \quad \text{(A.2.b)}$$

need to be assumed. In this appendix, the range of the required state variable for this approximation is determined. From (A.1), an approximation for $f(x)$ is provided.

The Simmons tunnel barrier model is appropriate when the state variable x is limited by a_{off} and a_{on}, i.e.,

$$a_{off} \le x \le a_{on}. \quad \text{(A.3)}$$

From the parameters in [18],

$$0 \le \frac{x - a_{off}}{w_c} \le \frac{a_{on} - a_{off}}{w_c} = \frac{1.8\mu - 1.2\mu}{107p} \approx \frac{600n}{0.1n} = 6\cdot 10^3 \Rightarrow$$
$$0 \le \frac{x - a_{off}}{w_c} \le 6\cdot 10^3$$
$$0 \le \frac{a_{on} - x}{w_c} \le 6\cdot 10^3 \quad \text{(A.4)}$$

Assume the maximum current in the device is 100 μA,

$$0 \le \frac{|i|}{b} \le \frac{100\mu}{500\mu} = \frac{1}{5}. \quad \text{(A.5)}$$

Assume that the value of the state variable is one of the effective boundaries a_{on} and a_{off},

$$x \approx a_{on} \Rightarrow \frac{1}{5} \ll \frac{x - a_{off}}{w_c} \le 6000$$
$$\Rightarrow \frac{x - a_{off}}{w_c} - \frac{|i|}{b} \approx \frac{x - a_{off}}{w_c}$$
$$x \approx a_{off} \Rightarrow \frac{1}{5} \ll \frac{a_{on} - x}{w_c} \le 6000$$
$$\Rightarrow \frac{a_{on} - x}{w_c} - \frac{|i|}{b} \approx \frac{a_{on} - x}{w_c} \quad \text{(A.6)}$$

To maintain the same approximation as in (A.6), it is sufficient to assume that the value of the expression in (A.5) is relatively small. Assume that one order of magnitude is sufficient for this assumption. The proper range of x can be determined as

$$\frac{|i|}{b} \le \frac{1}{5} \ll 2 \le \frac{x - a_{off}}{w_c} \Rightarrow 2w_c + a_{off} \le x \quad \text{(A.7)}$$

$$\frac{|i|}{b} \le \frac{1}{5} \ll 2 \le \frac{a_{on} - x}{w_c} \Rightarrow x \le a_{on} - 2w_c \quad \text{(A.8)}$$

For positive current, the derivative of x is positive and therefore the value of x is increasing. It is reasonable to assume (A.8). Similarly, for negative current, it is reasonable to assume (A.7). Under these assumptions, separation of variables can be achieved. See (A.9) at the top of the next page.

Based on the parameters in [18] and the exponential dependence, the exponential term is significantly greater than the second term,

$$x \approx a_{on}$$
$$\Rightarrow \exp\left(\frac{x - a_{off}}{w_c}\right) \approx \exp\left(\frac{a_{on} - a_{off}}{w_c}\right)$$
$$\approx \exp(6000) \gg \frac{x}{w_c} \approx \frac{a_{on}}{w_c} \approx 20$$
$$\Rightarrow \exp\left(\frac{x - a_{off}}{w_c}\right) \gg \frac{x}{w_c} \quad \text{(A.10)}$$

And similarly,

$$x \approx a_{off} \Rightarrow \exp\left(-\frac{x - a_{on}}{w_c}\right) \gg \frac{x}{w_c} \quad \text{(A.11)}$$

$$\frac{dx(t)}{dt} = \begin{cases} f_{off}\sinh\left(\frac{i}{i_{off}}\right)\exp\left[-\exp\left(\frac{x-a_{off}}{w_c}-\frac{|i|}{b}\right)-\frac{x}{w_c}\right], & i>0 \\ f_{on}\sinh\left(\frac{i}{i_{on}}\right)\exp\left[-\exp\left(-\frac{x-a_{on}}{w_c}-\frac{|i|}{b}\right)-\frac{x}{w_c}\right], & i<0 \end{cases}$$

$$\underset{\substack{i<0,\, x\approx a_{off} \\ i>0,\, x\approx a_{on}}}{\Rightarrow} \approx \begin{cases} f_{off}\sinh\left(\frac{i}{i_{off}}\right)\exp\left[-\exp\left(\frac{x-a_{off}}{w_c}\right)-\frac{x}{w_c}\right], & i>0 \\ f_{on}\sinh\left(\frac{i}{i_{on}}\right)\exp\left[-\exp\left(-\frac{x-a_{on}}{w_c}\right)-\frac{x}{w_c}\right], & i<0 \end{cases}$$

$$\Rightarrow \frac{dx(t)}{dt} = g(i)\cdot f(x)$$

$$\Rightarrow f(x) = \begin{cases} \exp\left[-\exp\left(\frac{x-a_{off}}{w_c}\right)-\frac{x}{w_c}\right], & i>0 \\ \exp\left[-\exp\left(-\frac{x-a_{on}}{w_c}\right)-\frac{x}{w_c}\right], & i<0 \end{cases} \tag{A.9}$$

From (A.10) and (A.11), the proposed window function is therefore

$$f_{on}(x) = \exp\left[-\exp\left(-\frac{x-a_{on}}{w_c}\right)\right] \tag{A.12}$$

$$f_{off}(x) = \exp\left[-\exp\left(\frac{x-a_{off}}{w_c}\right)\right] \tag{A.13}$$

ACKNOWLEDGMENT

The authors thank E. Yalon and O. Rottenstreich for their useful comments, and D. Belousov, S. Liman, E. Osherov, Z. Lati, D. Fliter, and K. Talisveyberg for their contributions to the SPICE and Verilog-A simulations.

REFERENCES

[1] L. O. Chua, "Memristor—The missing circuit element," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.
[2] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, Feb. 1976.
[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.
[4] D. Sacchetto, M. H. Ben-Jamaa, S. Carrara, G. DeMicheli, and Y. Leblebici, "Memristive devices fabricated with silicon nanowire schottky barrier transistors," in *Proc. IEEE Int. Symp. Circuits Syst.*, May/Jun. 2010, pp. 9–12.
[5] K. A. Campbell, A. Oblea, and A. Timilsina, "Compact method for modeling and simulation of memristor devices: Ion conductor chalcogenide-based memristor devices," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, Jun. 2010, pp. 1–4.
[6] X. Wang, Y. Chen, H. Xi, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE Electron Device Lett.*, vol. 30, no. 3, pp. 294–297, Mar. 2009.
[7] Y. Ho, G. M. Huang, and P. Li, "Nonvolatile memristor memory: Device characteristics and design implications," in *Proc. IEEE Int. Conf. Comput.-Aided Design*, Nov. 2009, pp. 485–490.
[8] A. Afifi, A. Ayatollahi, and F. Raissi, "Implementation of biologically plausible spiking neural network models on the memristor crossbar-based CMOS/Nano circuits," in *Proc. Eur. Conf. Circuit Theory Design*, Aug. 2009, pp. 563–566.
[9] Y. V. Pershin and M. Di Ventra, "Practical approach to programmable analog circuits with memristors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 8, pp. 1857–1864, Aug. 2010.
[10] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Mater. Sci. Process.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.
[11] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L. C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *J. Phys., Condens. Matter*, vol. 19, no. 16, pp. 1–13, Apr. 2007.
[12] Y. N. Joglekar and S. J. Wolf, "The elusive memristor: Properties of basic electrical circuits," *Eur. J. Phys.*, vol. 30, no. 4, pp. 661–675, Jul. 2009.
[13] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pt. 2, pp. 210–214, Jun. 2009.
[14] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A versatile memristor model with non-linear dopant kinetics," *IEEE Trans. Electron Devices*, vol. 58, no. 9, pp. 3099–3105, Sep. 2011.
[15] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, pp. 429–433, Jul. 2008.
[16] D. B. Strukov and R. S. Williams, "Exponential ionic drift: Fast switching and low volatility of thin-film memristors," *Appl. Phys. A, Mater. Sci. Process.*, vol. 94, no. 3, pp. 515–519, Mar. 2009.
[17] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *Proc. Int. Workshop Cell. Nanoscale Netw. Their Appl.*, Feb. 2010, pp. 1–4.
[18] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *J. Appl. Phys.*, vol. 106, no. 7, pp. 1–6, Oct. 2009.
[19] J. G. Simmons, "Generalized formula for the electric tunnel effect between similar electrodes separated by a thin insulating film," *J. Appl. Phys.*, vol. 34, no. 6, pp. 1793–1803, Jan. 1963.
[20] D. B. Strukov, J. L. Borghetti, and R. S. Williams, "Coupled ionic and electronic transport model of thin-film semiconductor memristive behavior," *Small*, vol. 5, no. 9, pp. 1058–1063, May 2009.
[21] H. Abdalla and M. D. Pickett, "SPICE modeling of memristors," in *IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 1832–1835.
[22] S. Benderli and T. A. Wey, "On SPICE macromodelling of TiO_2 memristors," *Electron. Lett.*, vol. 45, no. 7, pp. 377–379, Mar. 2009.
[23] S. Shin, K. Kim, and S.-M. Kang, "Compact models for memristors based on charge-flux constitutive relationships," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 4, pp. 590–598, Apr. 2010.
[24] T. Kawahara *et al.*, "2 Mb SPRAM (spin-transfer torque RAM) with Bit-by-Bit Bi-directional current write and parallelizing-direction current read," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 109–120, Jan. 2008.
[25] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE Electron Device Lett.*, vol. 32, no. 10, pp. 1436–1438, Oct. 2011.
[26] S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Verilog-A for memristors models," in *CCIT Tech. Rep. 801*, Dec. 2011.
[27] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY logic design procedure," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2011, pp. 142–147.
[28] E. Lehtonen, J. Poikonen, M. Laiho, and W. Lu, "Time-dependency of the threshold voltage in memristive devices," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 2245–2248.
[29] D. Biolek, Z. Biolek, and V. Biolkova, "Pinched hysteresis loops of ideal memristors, memcapacitors, and meminductors must be 'self-crossing'," *Electron. Lett.*, vol. 47, no. 25, pp. 1385–1387, Dec. 2011.
[30] L. O. Chua, "Resistance switching memories are memristors," *Appl. Phys. A, Mater. Sci. Process.*, vol. 102, no. 4, pp. 765–783, Mar. 2011.

Shahar Kvatinsky received his B.Sc. degree in computer engineering and applied physics, and an MBA degree from the Hebrew University of Jerusalem in 2009 and 2010, respectively. He is working toward the Ph.D. degree in the Electrical Engineering Department at the Technion—Israel Institute of Technology, Haifa.

Before his Ph.D. studies he worked for Intel as a circuit designer.

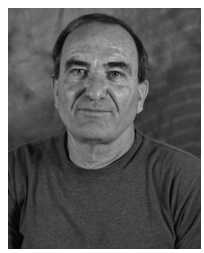
Eby G. Friedman (F'00) received the B.S. degree from Lafayette College, Easton, PA, in 1979, and the M.S. and Ph.D. degrees from the University of California, Irvine, in 1981 and 1989, respectively, all in electrical engineering.

From 1979 to 1991, he was with Hughes Aircraft Company, rising to the position of Manager of the Signal Processing Design and Test Department, responsible for the design and test of high performance digital and analog ICs. He has been with the Department of Electrical and Computer Engineering at the University of Rochester, Rochester, NY, since 1991, where he is a Distinguished Professor, and the Director of the High Performance VLSI/IC Design and Analysis Laboratory. He is also a Visiting Professor at the Technion—Israel Institute of Technology. His current research and teaching interests are in high performance synchronous digital and mixed-signal microelectronic design and analysis with application to high speed portable processors and low power wireless communications. He is the author of over 400 papers and book chapters, a dozen patents, and the author or editor of 15 books in the fields of high speed and low power CMOS design techniques, 3-D design methodologies, high speed interconnect, and the theory and application of synchronous clock and power distribution networks.

Dr. Friedman is the Regional Editor of the *Journal of Circuits, Systems and Computers*, a Member of the Editorial Boards of the *Analog Integrated Circuits and Signal Processing*, *Microelectronics Journal*, *Journal of Low Power Electronics*, *Journal of Low Power Electronics and Applications*, and the IEEE *Journal on Emerging and Selected Topics in Circuits and Systems*, Chair of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS steering committee, and a Member of the technical program committee of a number of conferences. He previously was the Editor-in-Chief of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, a Member of the Editorial Board of the PROCEEDINGS OF THE IEEE, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: ANALOG AND DIGITAL SIGNAL PROCESSING, and *Journal of Signal Processing Systems*, a Member of the Circuits and Systems (CAS) Society Board of Governors, Program and Technical chair of several IEEE conferences, and a recipient of the University of Rochester Graduate Teaching Award and a College of Engineering Teaching Excellence Award. He is a Senior Fulbright Fellow.

Avinoam Kolodny (SM'11) received his Ph.D. degree in microelectronics from Technion—Israel Institute of Technology, Haifa, in 1980.

He joined Intel Corporation, where he was engaged in research and development in the areas of device physics, VLSI circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Electrical Engineering at the Technion since 2000. His current research is focused primarily on interconnects in VLSI systems, at both physical and architectural levels.



Uri C. Weiser (F'02) received his B.S. and M.S. degrees in EE from Technion—Israel Institute of Technology, Haifa, and a Ph.D degree in CS from the University of Utah, Salt Lake City.

He is a visiting Professor at the Electrical Engineering Department, Technion IIT, and acts as an advisor at numerous startups. He worked at Intel from 1988 to 2006. At Intel, he initiated the definition of the first Pentium® processor, drove the definition of Intel's MMX™ technology, invented (with A. Peleg) the Trace Cache, co-managed and established the Intel Microprocessor Design Center at Austin, TX, and later initiated an Advanced Media applications research activity. Prior to his career at Intel, he worked for the Israeli Department of Defense as a research and system engineer and later with the National Semiconductor Design Center in Israel, where he led the design of the NS32532 microprocessor.

Dr. Weiser was appointed Intel Fellow in 1996, and in 2005 he became an ACM Fellow. He was an Associate Editor of *IEEEMicro Magazine* (1992–2004) and was Associate Editor of *Computer Architecture Letters*.

Models of Memristors for SPICE Simulations

Shahar Kvatinsky, Keren Talisveyberg, Dmitry Fliter, Avinoam Kolodny, and Uri C. Weiser
Department of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 32000, ISRAEL
skva@tx.technion.ac.il

Eby G. Friedman
Department of Electrical and Computer Engineering
University of Rochester
Rochester, New York 14627, USA

***Abstract*— Memristors are novel devices which can be used in applications such as memory, logic, analog circuits, and neuromorphic systems. Several memristor technologies have been developed such as ReRAM (Resistive RAM), MRAM (Magnetoresistance RAM), and PCM (Phase Change Memory). To design circuits with memristors, the behavior of the memristor needs to be described by a mathematical model. While the model for memristors should be sufficiently accurate as compared to the behavior of physical devices, the model must also be computationally efficient. Several models for memristors have been proposed – the linear ion drift model, the nonlinear ion drift model, the Simmons tunnel barrier model, and the ThrEshold Adaptive Memristor (TEAM) model. In this paper, the different memristor models are described and a Verilog-A implementation for these models, including the relevant window functions, are presented. These models are suitable for EDA tools such as SPICE.**

***Index Terms*— memristor, memristive systems, SPICE, Verilog-A, TEAM.**

I. Introduction

Memristors are passive two-port elements with variable resistance (also known as a memristance) [1]. Changes in the memristance depend upon the history of the device (e.g., the memristance may depend on the total charge passed through the device, or alternatively, on the integral over time of the applied voltage between the ports of the device). Memristive systems [2] are an extension to memristors, where a current-controlled time-invariant memristive device is represented by

$$\frac{dw}{dt} = f(w,i), \tag{1}$$

$$v(t) = R(w,i)\cdot i(t), \tag{2}$$

where w is an internal state variable, i(t) is the current of the memristive device, $v(t)$ is the voltage across the memristive device, $R(w, i)$ is the memristance, and t is time. The terms memristor and memristive systems are often used interchangeably to describe memristive devices.

Memristors can be used in applications such as memory, logic, analog circuits, and neuromorphic systems. A memristor offers several advantages as compared to standard memory technologies: nonvolatility, good scalability, effectively no leakage current, and compatibility with CMOS technology, both electrically and in terms of manufacturing. Several memristor technologies have been developed such as ReRAM (Resistive RAM), MRAM (Magnetoresistance RAM), and PCM (Phase Change Memory).

To design circuits with memristors, the behavior of the memristor needs to be described by a mathematical model. While the model for memristors should be sufficiently accurate as compared to the behavior of physical devices, it must also be computationally efficient. It is also desirable for the model to be simple, intuitive, and closed-form, as well as general so that it can be tuned to suit different technologies of memristors. Several memristor models have been proposed: the linear ion drift model, the nonlinear ion drift model, the Simmons tunnel barrier model, and the ThrEshold Adaptive Memristor (TEAM) model. In this paper, the different memristor models are described and a Verilog-A code for these models and the relevant window functions are presented. These models are suitable for EDA tools such as SPICE.

II. Memristor Models

All of the memristor models which have been implemented in the Verilog-A model are described in [3]. In this paper, only a brief description of these models is provided. The basic equations and the main characteristics of the memristor models are listed in Table 1. A user manual to this Verilog-A model is provided in [4].

A. Linear Ion Drift Model

In the linear ion drift model [5], two resistors are connected in series, one resistor represents the high concentration of dopants region (high conductance) and the second resistor represents the oxide region (low conductance). A linear ion drift in a uniform field is also assumed, where the ions have equal average ion mobility μ_V. This model exhibits the definition of the original memristor in [1], but is inaccurate as compared to physical memristive devices.

B. Nonlinear Ion Drift Model

In the nonlinear ion drift model [6], a voltage-controlled memristor exhibiting a nonlinear dependence between the voltage and the internal state derivative is assumed. In this model, the state variable w is a normalized parameter within the interval [0, 1]. This model also assumes an asymmetric switching behavior.

C. Simmons Tunnel Barrier Model

The Simmons tunnel barrier model [7] assumes nonlinear and asymmetric switching behavior due to an exponential dependence of the movement of the ionized dopants, namely, changes in the internal state variable. In this model, rather than two resistors in series as in the linear drift model, there is a resistor in series with an electron tunnel barrier. The state variable *x* is the Simmons tunnel barrier width.

D. ThrEshold Adaptive Memristor (TEAM) Model

The TEAM model [3] is a general memristor model. In this model, a current threshold and tunable nonlinear (polynomial) dependence between the current and the derivative of the internal state variable are assumed. The current-voltage relationship can be in a linear or an exponential manner. It is possible to fit the TEAM model to the Simmons tunnel barrier model or to any different memristor model and gain a more efficient computational time with sufficient accuracy.

III. Window Functions

To maintain the physical bounds of the device and add nonlinear behavior close to these physical bounds, several window functions are implemented in the Verilog-A model. These window functions are: Jogelkar [8], Biolek [9], Prodromakis [10], and TEAM (named Kvatinsky in the Verilog-A model). The window functions and main properties are listed in Table 2.

IV. Verilog-A Code

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

// define meter units for w parameter nature distance
    access = Metr;
    units = "m";
    abstol = 0.01n;
endnature
    discipline Distance
    potential distance;
enddiscipline

module Memristor(p, n,w_position);
    input p;//positive pin
    output n;//negative pin
    output w_position;// w-width pin
    electrical p, n,gnd;
    Distance w_position;
    ground gnd;
    parameter real model = 0;    // define the model 0 - Linear
    // Ion Drift ; 1 - Simmons Tunnel Barrier; 2 - Team model;
   // 3- Nonlinear Ion Drift model
   parameter real window_type=0;   // define the window type :
   // 0 - No window; 1 - Jogelkar window ; 2 - Biolek window ;
   // 3 - Prodromakis window ; 4- Kvatinsky window (Team
   //                    model                    only)
  parameter real dt=0; // user must specify dt same as max step
    // size in transient analysis & must be at least 3 orders
    // smaller than T period of the source
    parameter real init_state=0.5; // the initial state condition
    // [0:1]

//////////////////////Linear Ion Drift model ////////////////////////////////
//parameters definitions and default values for linear model
    parameter real Roff = 200000;
    parameter real Ron = 100;
    parameter real D = 3n;
    parameter real uv = 1e-15;
    parameter real w_multiplied = 1e8;  // transformation factor
    // for w/X width in meter units
    parameter real p_coeff = 2; // Windowing function
    // coefficient
    parameter real J = 1;  // for prodromakis Window function
    parameter real p_window_noise=1e-18; // provoke the w
    // width not to get stuck at 0 or D with p window
    parameter real treshhold_voltage=0;

    // local variables
    real w;
    real dwdt;
    real w_last;
    real R;
    real sign_multply;
    real stp_multply;
    real first_iteration;

////////////////////// Simmons Tunnel Barrier model ////////////////////
 //parameters definitions and default values
    parameter real f_off = 3.5e-6;
    parameter real f_on = 40e-6;
    parameter real i_off = 115e-6;
    parameter real i_on = 8.9e-6;
    parameter real x_c = 107e-12;
    parameter real b = 500e-6;
    parameter real a_on = 2e-9;
    parameter real a_off = 1.2e-9;

    // local variables
    real x;
    real dxdt;
    real x_last;

////////////////////////////TEAM model////////////////////
    parameter real K_on=-8e-13;
    parameter real K_off=8e-13;
    parameter real Alpha_on=3;
    parameter real Alpha_off=3;
    parameter real IV_relation=0; // IV_relation=0 means linear
    // V=IR. IV_relation=1 means nonlinear V=I*exp{..}
    parameter real x_on=0;
    parameter real x_off=3e-09;  // equals D
    // local variables
    real lambda;
```

```
////////////////////////////Nonlinear Ion Drift model/////////////////////
   parameter real alpha = 2;
   parameter real beta  = 9;
   parameter real c     = 0.01;
   parameter real g     = 4;
   parameter real N     = 14;
   parameter real q     = 13;
   parameter real a     = 4;

 analog function integer sign; //Sign function for Constant
 // edge cases
    real arg; input arg;
    sign = (arg >= 0 ? 1 : -1 );
 endfunction
 analog function integer stp;            //Stp function
    real arg; input arg;
    stp = (arg >= 0 ? 1 : 0 );
 endfunction

 ////////////////////////MAIN ////////////////////////////////
 analog begin
           if(first_iteration==0) begin
           w_last=init_state*D; // if this is the first iteration,
          // start with w_init
           x_last=init_state*D;         // if this is the first
          // iteration, start with x_init
           end

//////////////////////Linear Ion Drift model ////////////////////////////////
if (model==0) begin // Linear Ion Drift model
           dwdt =(uv*Ron/D)*I(p,n);
  //change the w width only if the threshhold_voltage permits!
     if(abs(I(p,n))<treshhold_voltage/R) begin
          w=w_last;
                dwdt=0;
     end
     if ((window_type==0)|| (window_type==4)) begin // No
     // window
      w=dwdt*dt+w_last;
          end // No window
          if (window_type==1) begin // Jogelkar window
                if (sign(I(p,n))==1) begin
                      sign_multply=0;
                      if(w==0) begin
                      sign_multply=1;
                      end
                end
                if (sign(I(p,n))==-1) begin
                sign_multply=0;
                     if(w==D) begin
                     sign_multply=-1;
                     end
                end
          w=dwdt*dt*(1-pow(2*w/D-
         1,2*p_coeff))+w_last+sign_multply*p_window_noise;
          end // Jogelkar window
          if (window_type==2) begin // Biolek window
                if (stp(-I(p,n))==1) begin
                      stp_multply=1;
                end
                if (stp(-I(p,n))==0) begin
                stp_multply=0;
               end
          w=dwdt*dt*(1-pow(w/D-
             stp_multply,2*p_coeff))+w_last;
          end  // Biolek window
          if (window_type==3) begin // Prodromakis window
                if (sign(I(p,n))==1) begin
                      sign_multply=0;
                      if(w==0) begin
                      sign_multply=1;
                      end
                end
                if (sign(I(p,n))==-1) begin
                sign_multply=0;
                     if(w==D) begin
                    sign_multply=-1;
                     end
                end
           w=dwdt*dt*J*(1-pow(pow(w/D-
           0.5,2)+0.75,p_coeff))+  w_last  +  sign_multply  *
           p_window_noise;
          end  // Prodromakis window
          if (w>=D) begin
                w=D;
               dwdt=0;
          end
          if (w<=0) begin
                w=0;
               dwdt=0;
          end

       //update the output ports(pins)
       R=Ron*w/D+Roff*(1-w/D);
       w_last=w;
       Metr(w_position) <+ w*w_multiplied;
       V(p,n) <+ (Ron*w/D+Roff*(1-w/D))*I(p,n);
       first_iteration=1;
end // end Linear Ion Drift model

//////////////////////Simmons Tunnel Barrier model//////////////////////
if (model==1) begin // Simmons Tunnel Barrier model
           if (sign(I(p,n))==1) begin
             dxdt      =f_off*sinh(I(p,n)/i_off)*exp(-exp((x_last-
             a_off)/x_c-abs(I(p,n)/b))-x_last/x_c);
           end
           if (sign(I(p,n))==-1) begin
             dxdt    =    f_on*sinh(I(p,n)/i_on)*exp(-exp((a_on-
             x_last)/x_c-abs(I(p,n)/b))-x_last/x_c);
           end
        x=x_last+dt*dxdt;
           if (x>=D) begin
                x=D;
```

```
            dxdt=0;
          end
          if (x<=0) begin
            x=0;
            dxdt=0;
          end
      //update the output ports(pins)
      R=Ron*(1-x/D)+Roff*x/D;
      x_last=x;
      Metr(w_position) <+ x/D;
      V(p,n) <+ (Ron*(1-x/D)+Roff*x/D)*I(p,n);
      first_iteration=1;
end // end Simmons Tunnel Barrier model

////////////////////////////////TEAM model/////////////////////////////////
if (model==2) begin // Team model
          if (I(p,n) >= i_off) begin
          dxdt =K_off*pow((I(p,n)/i_off-1),Alpha_off);
          end
          if (I(p,n) <= i_on) begin
          dxdt =K_on*pow((I(p,n)/i_on-1),Alpha_on);
          end
          if ((i_on<I(p,n)) && (I(p,n)<i_off)) begin
          dxdt=0;
          end
          if (window_type==0) begin // No window
          x=x_last+dt*dxdt;
          end // No window
          if (window_type==1) begin // Jogelkar window
                    x=x_last+dt*dxdt*(1-pow((2*x_last/D-
                  1),(2*p_coeff)));
          end // Jogelkar window
          if (window_type==2) begin // Biolek window
                    if (stp(-I(p,n))==1) begin
                              stp_multply=1;
                    end
                    if (stp(-I(p,n))==0) begin
                              stp_multply=0;
                   end
                  x=x_last+dt*dxdt*(1-pow((x_last/D-
                  stp_multply),(2*p_coeff)));
          end  // Biolek window
          if (window_type==3) begin // Prodromakis window
                    x=x_last+dt*dxdt*J*(1-
                    pow((pow((x_last/D-0.5),2)+0.75),p_coeff));
          end  // Prodromakis window
          if (window_type==4) begin //Kvatinsky window
                    if (I(p,n) >= 0) begin
                         x=x_last+dt*dxdt*exp(-exp((x_last-
                         a_off)/x_c));
                    end
                     if (I(p,n) < 0) begin
                        x      =      x_last+dt*dxdt*exp(-exp((a_on-
                        x_last)/x_c));
                     end
      end // Kvatinsky window
          if (x>=D) begin
            dxdt=0;
            x=D;
          end
          if (x<=0) begin
            dxdt=0;
            x=0;
          end
          lambda = ln(Roff/Ron);
          //update the output ports(pins)
          x_last=x;
          Metr(w_position) <+ x/D;
          if (IV_relation==1) begin
            V(p,n)          <+          Ron*I(p,n)*exp(lambda*(x-
            x_on)/(x_off-x_on));
          end
          else if (IV_relation==0) begin
            V(p,n) <+ (Roff*x/D+Ron*(1-x/D))*I(p,n);
          end
          first_iteration=1;
end // end TEAM model

////////////////////////////Nonlinear Ion Drift model/////////////////////
if (model==3) begin // Nonlinear Ion Drift model
          if (first_iteration==0) begin
            w_last=init_state;
          end
          dwdt = a*pow(V(p,n),q);
          if  ((window_type==0)  ||  (window_type==4))  begin
          // No window
            w=w_last+dt*dwdt;
          end // No window
          if (window_type==1) begin // Jogelkar window
            w=w_last+dt*dwdt*(1-pow((2*w_last-1)  ,  (2  *
            p_coeff)));
          end // Jogelkar window
          if (window_type==2) begin // Biolek window
                    if (stp(-V(p,n))==1) begin
                              stp_multply=1;
                    end
                    if (stp(-V(p,n))==0) begin
                    stp_multply=0;
                    end
                   w=w_last+dt*dwdt*(1-pow((w_last-
                   stp_multply),(2*p_coeff)));
          end // Biolek window
          if (window_type==3) begin // Prodromakis window
                w=w_last+dt*dwdt*J*(1-pow((pow((w_last-
                0.5),2)+0.75),p_coeff));
          end  // Prodromakis window
          if (w>=1) begin
                    w=1;
                    dwdt=0;
          end
          if (w<=0) begin
                    w=0;
                    dwdt=0;
          end
```

```
  //change the w width only if the threshhold_voltage permits!
        if(abs(V(p,n))<treshhold_voltage) begin
        w=w_last;
        end

    //update the output ports(pins)
         w_last=w;
         Metr(w_position) <+ w;
         I(p,n) <+ pow(w,N) *beta *sinh(alpha*V(p,n)) +c*
        (exp(g*V(p,n))-1);
        first_iteration=1;
    end // end Nonlinear Ion Drift model
 end   // end analog
endmodule
```

V. Conclusions

A Verilog-A code that contains several models, useful for design in memristor-based circuits, is presented in this paper, as well as relevant window functions. This Verilog-A model can be used by circuit designers, since it is easy to use, contains several mathematical models, the parameters of already existing models can be easily changed, as well as additional mathematical models can be added.

Acknowledgment

This work was partially supported by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, and by Intel grant no. 864-737-13.

References

[1] L. O. Chua, “Memristor – the Missing Circuit Element,” *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[2] L.O. Chua and S.M. Kang, “Memristive Devices and Systems,” *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209-223, February 1976.

[3] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2012 (in press).

[4] http://memristor.shorturl.com

[5] D. B. Strukov, G. S.Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found,” *Nature*, Vol. 453, pp. 80-83, May 2008.

[6] E. Lehtonen and M. Laiho, "CNN Using Memristors for Neighborhood Connections," *Proceedings of the International Workshop on Cellular Nanoscale Networks and their Applications*, pp. 1-4, February 2010.

[7] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching Dynamics in Titanium Dioxide Memristive Devices," *Journal of Applied Physics*, Vol. 106, No. 7, pp. 1-6, October 2009.

[8] Y. N. Joglekar and S. J. Wolf, “The Elusive Memristor: Properties of Basic Electrical Circuits,” *European Journal of Physics*, Vol. 30, No. 4, pp. 661-675, July 2009.

[9] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, Vol. 18, No. 2, Part 2, pp. 210-214, June 2009.

[10] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, “A Versatile Memristor Model with Non-linear Dopant Kinetics,” *IEEE Transactions on Electron Devices*, Vol. 58, No. 9, pp. 3099-3105, September 2011.

Table 1. The Different Memristor Models (Further description in [2])

Model	Linear ion drift [5]	Nonlinear ion drift [6]	Simmons tunneling barrier [7]	TEAM [3]
State variable	$0 \le w \le D$ Doped region physical width	$0 \le w \le 1$ Doped region normalized width	$a_{off} \le x \le a_{on}$ Undoped region width	$x_{on} \le x \le x_{off}$ Undoped region width
Control mechanism	Current controlled	Voltage controlled	Current controlled	Current controlled
Threshold	None	None	None	i_{on}, i_{off}

Table 2. Comparison of Different Window Functions

Function	Jogelkar [8]	Biolek [9]	Prodromakis [10]	TEAM [3]
$f(x)/f(w)$	$f(w) = 1-(2w/D-1)^{2p}$	$f(w) = 1-(w/D-stp(-i))^{2p}$	$f(w)=j(1-[(w-0.5)^2+0.75]^p)$	$f_{on,off}=\exp[-\exp(\lvert x-x_{on,off}\rvert/w_c)]$
Fits memristor model	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	Linear/nonlinear ion drift/TEAM	TEAM for Simmons tunneling barrier fitting

The Desired Memristor for Circuit Designers

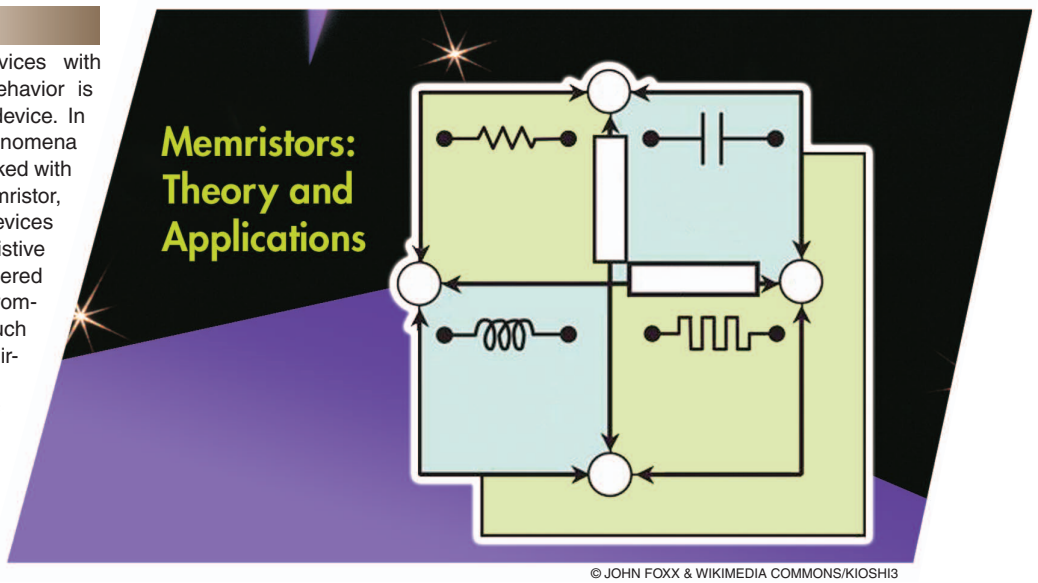
Shahar Kvatinsky, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser

Abstract

Memristors are two-terminal devices with varying resistance, where the behavior is dependent on the history of the device. In recent years, different physical phenomena of resistive switching have been linked with the theoretical concept of a memristor, and several emerging memory devices (e.g., Phase Change Memory, Resistive RAM, STT-MRAM) are now considered as memristors. Memristors hold promise for use in diverse applications such as memory, digital logic, analog circuits, and neuromorphic systems.

Important characteristics of memristors include high speed, low power, good scalability, data retention, endurance, and compatibility with conventional CMOS in terms of manufacturing and operating voltages. One interesting property of some memristors is a nonlinear response to current or voltage. Nonlinear memristors exhibit a current or voltage threshold, such that the resistance is affected only by currents or voltages which exceed the threshold, while the resistance of a linear memristor changes with small perturbations in device current.

Different applications exploit different characteristics of a memristor. In this article, the desired characteristics for different applications are presented from the viewpoint of an integrated circuit designer. Understanding the desired characteristics for different applications can assist device and material engineers in providing the appropriate behavior when developing memristive devices, thereby optimizing these devices for different applications.



I. Introduction

Memristors have many different facets. A memristor can be considered as the theoretical missing fundamental element originally proposed by Leon Chua in 1971 [1]. This theoretical device is a resistor with varying resistance, where the resistance changes according to the charge passed through the memristor over its entire history. Chua extended the theory of memristors to 'memristive devices' in 1976 with his student, Steve (Sung Mo) Kang [2]. A memristive device is basically any resistor with a resistance that only changes due to the voltage across the device or, alternatively, the current flowing through the device. Since the resistance does not change when there is no voltage applied across the device, memristive devices are nonvolatile. It is acceptable to use the term 'memristor' to describe a 'memristive device.'

Since Hewlett Packard Laboratories announced the fabrication of a working memristor by electrical conduction in titanium oxide (TiO_2) in 2008 [3], it has become popular to link different physical phenomena of resistive switching with the term memristor. These devices include a large variety of oxides, also named Resistive RAM (RRAM). Additional emerging memory devices (e.g., Phase Change Memory and STT-MRAM) may also be considered as memristors since these devices are basically nonvolatile two-terminal devices with varying

Digital Object Identifier 10.1109/MCAS.2013.2256257
Date of publication: 22 May 2013

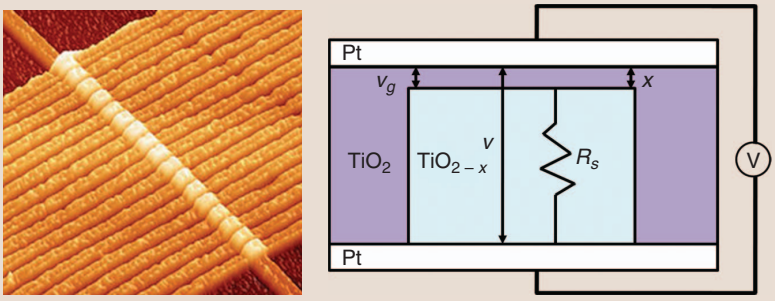


Figure 1. (a) An array of 17 purpose-built oxygen-depleted titanium dioxide (TiO_2) memristors built at HP Labs, imaged by an atomic force microscope. The wires are about 50 nm wide (credit: J.J. Yang, HP Labs), and (b) a physical structure model of a TiO_2 memristor.

resistance. In this article, memristors are considered in their broadest meaning—any two-terminal device with memory capability, which is represented by a varying resistance. An array of TiO_2 memristors and a schematic of the physical structure of a single device are shown in Figure 1.

II. The Desired Memristor

Using memristors as storage elements in a memory is an obvious choice. Actually, most emerging memory technologies, which are considered as potential replacements for Flash, DRAM, and SRAM, are based on memristors. These technologies are somewhat immature and are not yet fully commercialized. Toshiba and Sandisk are currently sampling 4 GB RRAM memory circuits [4], Micron and Samsung are selling 16 MB PCM [5], and Everspin recently debuted an 8 MB STT-MRAM [6]. Memristors are, however, much more than memory devices. The ability to control and modify their current-voltage characteristics can be utilized for performing a variety of computational operations. Memristors hold promise for use in diverse applications such as digital and analog circuits, and neuromorphic systems.

Since memristor technology is currently immature, standardization of the characteristics of memristors remains to be done. The desired characteristics may differ for different applications. In this article, the desired characteristics of memristors are described for different applications from the viewpoint of an integrated circuit designer. Understanding the desired characteristics for different applications can assist device and material engineers in providing the appropriate behavior when developing memristive devices, thereby optimizing these devices for different applications.

III. Memory

The speed, power consumption, data retention, and endurance of memristors are better than Flash memory for all emerging memristive technologies, and are comparable to DRAM and SRAM for certain memristive technologies. Speed is determined by the write time which currently varies from tens of nanoseconds (PCM) to hundreds of picoseconds (RRAM). Endurance is determined by the number of writes to a device without affecting the stored data, and currently varies from hundreds of millions of writes (PCM) to an unlimited number of writes (STT-MRAM). All emerging memory technologies satisfy the industrial standard of ten year data retention. STT-MRAM, however, still suffers degradation in data retention for technologies below 45 nm. A summary of the required characteristics for memory applications is listed in Table 1.

One interesting property of some memristors is a nonlinear response to current or voltage. Nonlinear memristors exhibit a current or voltage threshold, such that the resistance is not affected by relatively small currents or voltages, while the resistance of a linear memristor will change due to any change in device current. In the original publication by Hewlett Packard in 2008 [3], a linear memristor was presented. Practical memristors, however, seem to behave nonlinearly, although the nonlinearity varies for different materials and technologies. Current–voltage curves of linear and nonlinear memristor are shown in Figure 2.

Due to excellent scalability and fast speed, memristors are a potential replacement for Flash memory in SSD, which requires dense memory, as well as DRAM and SRAM for main memory and cache memory, which require relatively fast memory with unlimited writes. Memristors therefore provide an opportunity for 'universal memory'—a single technology for all memory hierarchies. Memristors are by their definition nonvolatile devices. Using memristors within caches and main memory will make these memories nonvolatile, dramatically changing the manner in which these memories

Shahar Kvatinsky, Avinoam Kolodny, and Uri C. Weiser are with Technion—Israel Institute of Technology, Haifa 3200, Israel. Eby G. Friedman is with the University of Rochester, Rochester, New York, 14627 USA.

are applied in modern computing systems.

Memory is an analog circuit behaving digitally, in which the resistance of the memristor typically represents a binary value. A low resistance is typically considered as a 'logical one' and a high resistance is treated as a 'logical zero.' A high ratio between the high and low resistance (usually named, respectively, R_{OFF} and R_{ON}) is therefore desirable. It is also desirable to provide a nondestructive read mechanism, but the read operation in memristors may induce drift in the stored state. The drift requires occasionally refreshing the memory. The device design process should therefore consider the trade off between speed and robustness due to this state drift phenomenon. A preferred memristor would therefore be highly nonlinear, with a well-defined and abrupt threshold between the two distinct states.

Table 1.
Requirements of memristors for memory [11].

	Speed (Write Time) [Seconds]	Endurance [# Writes]	Energy Per Bit [Joule]	Nonvolatility
Storage (flash replacement)	0.1 to 10 μ	10^5	10 n	Yes
Main Memory (DRAM replacement)	10 n	$> 10^{15}$	5 p	No
Cache (SRAM replacement)	0.3 to 1 n	$> 10^{15}$	5 p	No

In memory applications, it is also possible to write more than one bit into a single memristor if the resistance of the device can be quantized into multiple levels. The difference among the different data must be carefully determined. To successfully store more than one bit within a memristor, it is crucial to maintain a high ratio between R_{OFF} and R_{ON} to provide a wide range of resistance. It is also preferable that a linear memristive device successfully write the desired value with similar write pulses, or, alternatively, that a write mechanism allows a different and distinct write operation for different data. In PCM, for example, the write operation uses a different magnitude and duration of applied current to write different data, as depicted in Figure 3.

IV. Computational Logic with Memristors

Another application of memristors is computational logic, where memristors are used as logic gates. Several different logic families have been developed that use memristors as fundamental elements within logic gates. In certain logic families, the logical state is represented as a resistance, as in memory, and the result of the logical operation is also stored as a resistance in a memristor. These logic families can therefore be used for *logic within memory*, and require similar memristor characteristics as in memory, namely, nonlinear memristors with well-defined thresholds are preferable. An example of these logic families is material implication (IMPLY) [7], as shown in Figure 4. In other logic families, the logical state is represented as a voltage level, as in CMOS logic. These logic families are useful for *hybrid CMOS-memristor*

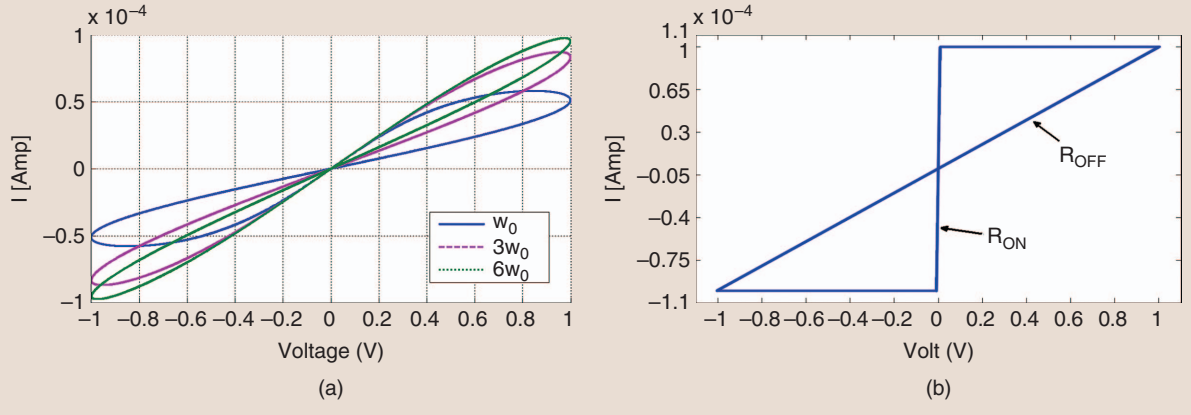


Figure 2. Current–voltage curve in response to sinusoidal input for (a) a linear memristor (voltage input) and (b) nonlinear memristor (current input). ω_0 is a frequency of the input waveform. The maximum and minimum resistance of the memristor, respectively, R_{off} and R_{on}, are marked in the nonlinear memristor curve.

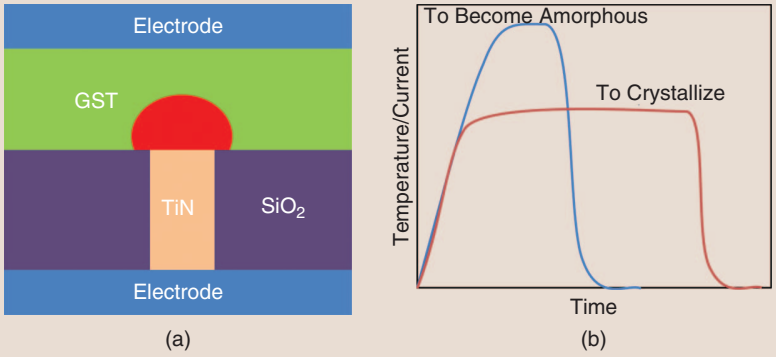


Figure 3. Phase change memory (PCM) (a) physical structure. A resistor made of TiN acts as a heater and heats the active area (marked in red). The active area heats the GST (chalcogenide glass), which changes its phase between crystalline and amorphous states. Crystalline state has better conductivity than amorphous state. (b) The write operation is done by flowing current in different shapes where high current (temperature) for a short period changes the phase to amorphous and relatively low current for a long period changes the phase to crystalline.

(MRL) [8], as shown in Figure 5. It is also possible to use memristors as configurable switches in PLA and FPGA [9], as shown in Figure 6. For these applications, the memristors replace the standard programmable switches, commonly placed within the FPGA as CMOS switch boxes. High and low resistances are treated, respectively, as an 'open' and 'closed' switch. In these applications, the configuration phase is separate from the operation. The resistance of the memristors therefore does not change during operation and a nonlinear memristor with a threshold is necessary. A significant ratio between the high and low resistance is also highly desirable.

logic, where the critical characteristics of the memristors are their high density and compatibility with standard CMOS, both in fabrication and voltage levels. These logic families increase the logic density, where, for the same area, the number of logic gates is significantly higher. For these logic families, a linear memristor is preferable to reduce power consumption and delay. An example of these logic families is Memristor Ratioed Logic

V. Analog Circuits and Neuromorphic Systems

In applications using analog circuits and neuromorphic systems (electronic circuits that mimic the brain), the resistance typically requires a continuous value. Memristors can be used as configurable devices where the resistance of the device is initialized by a specific procedure, different from typical circuit operation [10]. During regular circuit operation, the memristor

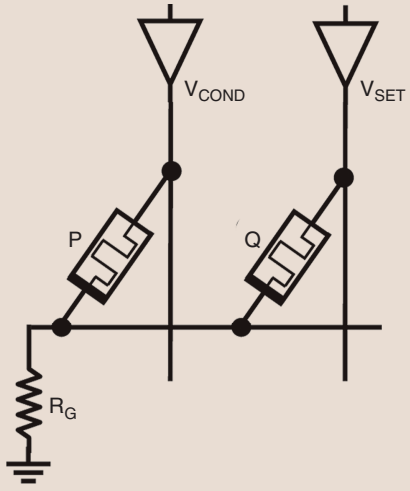


Figure 4. Schematic of a memristor-based material implication (IMPLY) logic gate. IMPLY gate consists of two memristors and a resistor. The memristors can be part of a memristor-based crossbar used for memory. The input and output variables of the IMPLY logic gate are the stored logical state of the memristors, represented by their initial and final resistance, where high and low resistance are considered, respectively, as logical zero and one.

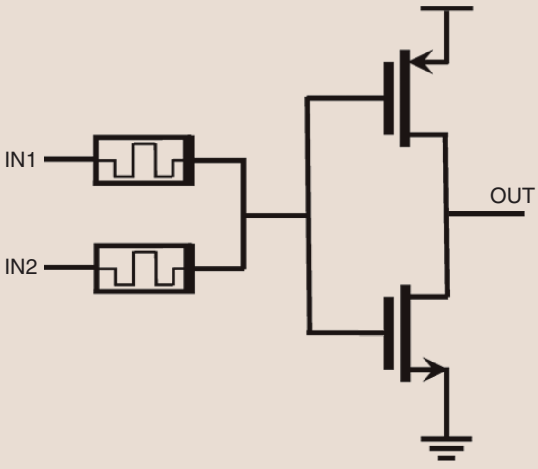


Figure 5. An example of hybrid memristor-CMOS logic—memristor ratioed logic (MRL). An MRL NAND logic gate consists of two memristors and two CMOS transistors. The memristors act as a logical AND gate and are connected to a CMOS-based inverter. The logical input and output variables are represented by voltages, as in conventional CMOS logic.

behaves as a simple resistor. The properties of the circuit can be tuned. A configurable amplifier is shown in Figure 7, where the gain and bandwidth of an amplifier vary due to the configurable resistance. In these applications, it is desirable for the memristor to behave as a nonlinear nondestructive device, similar to the read mechanism in digital applications. Memristors can also be used as computational elements in analog circuits, such as analog counters and sensors. In these circuits, it is desirable for the memristor to maintain a linear behavior, where the local current changes the resistance of the memristor.

In neural networks, memristors mimic the role of synapses, such that each device may interact with other devices throughout the system. Several models exist for using memristors in neuromorphic systems. Usually, machine learning algorithms are executed in these systems. A threshold is useful to disable the learning operation. During the learning operation, the resistance of the memristor is changed based on the input of the system, usually a voltage pulse. It is desirable for the same input to change the resistance of the memristor the same every time. Nonlinear memristors require the change in resistance to be significantly different for the same input with a different initial resistance, greatly complicating the learning process.

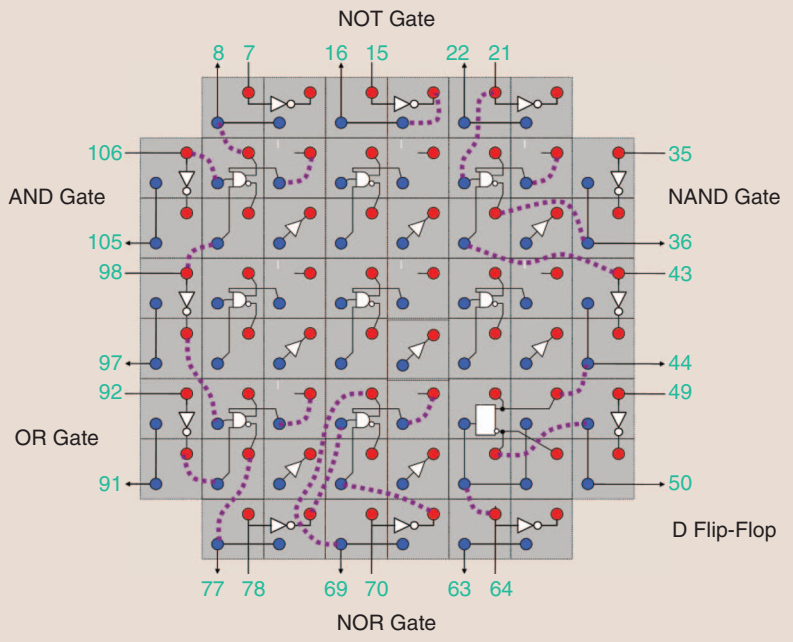


Figure 6. An example of a memristor-based FPGA. The memristors act as configurable switches to connect or disconnect CMOS logic gates. (This figure is taken from [9].)

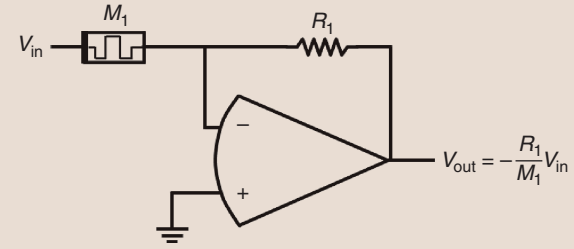


Figure 7. Schematic of a configurable amplifier. The gain of the amplifier is the ratio between the resistors and can be tuned to a desired value. The memristor is programmed to a desired resistance prior to the operation of the amplifier.

VI. Conclusions

In summary, memristors provide an inspiring variety of opportunities for electronics. Memristor technology is still immature and the device characteristics can vary a great deal. However, significant focus within academia and industry is currently taking place to develop and commercialize this exciting new technology. In this article, certain desirable characteristics of memristors are described for an assortment of applications. It is intended that device and material engineers will consider the requirements for these devices from the point of view of an integrated circuit designer, and develop devices suitable for specific applications, opening a new era of memory intensive computing.

Shahar Kvatinsky is a Ph.D. candidate at the electrical engineering department at the Technion—Israel Institute of Technology. He received his B.Sc. in computer engineering and applied physics, and an MBA at 2009 and 2010, respectively, both from the Hebrew University of Jerusalem. Prior to his Ph.D. studies he worked for Intel as a circuit designer.

Eby G. Friedman received the B.S. degree from Lafayette College in 1979, and the M.S. and Ph.D. degrees from the University of California, Irvine, in 1981 and 1989, respectively, all in electrical engineering. From 1979 to 1991, he was with Hughes Aircraft Company,

rising to the position of manager of the Signal Processing Design and Test Department, responsible for the design and test of high performance digital and analog ICs. He has been with the Department of Electrical and Computer Engineering at the University of Rochester since 1991, where he is a Distinguished Professor, and the Director of the High Performance VLSI/IC Design and Analysis Laboratory. He is also a Visiting Professor at the Technion—Israel Institute of Technology. His current research and teaching interests are in high performance synchronous digital and mixed-signal microelectronic design and analysis with application to high speed portable processors and low power wireless communications. He is the author of over 400 papers and book chapters, 11 patents, and the author or editor of 15 books in the fields of high speed and low power CMOS design techniques, 3-D design methodologies, high speed interconnect, and the theory and application of synchronous clock and power distribution networks. Dr. Friedman is the Regional Editor of the *Journal of Circuits, Systems and Computers*, a Member of the editorial boards of the *Analog Integrated Circuits and Signal Processing*, *Microelectronics Journal*, *Journal of Low Power Electronics*, *Journal of Low Power Electronics and Applications*, and *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Chair of the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* steering committee, and a Member of the technical program committee of a number of conferences. He previously was the Editor-in-Chief of the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, a Member of the editorial board of the *Proceedings of the IEEE*, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, and *Journal of Signal Processing Systems*, a Member of the Circuits and Systems (CAS) Society Board of Governors, Program and Technical chair of several IEEE conferences, and a recipient of the University of Rochester Graduate Teaching Award and a College of Engineering Teaching Excellence Award. Dr. Friedman is a Senior Fulbright Fellow and an IEEE Fellow.

Avinoam Kolodny received his doctorate in microelectronics from Technion—Israel Institute of Technology in 1980. He joined Intel Corporation, where he was engaged in research and development in the areas of device physics, VLSI circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Electrical Engineering at the Technion since 2000. His current research is focused primarily on interconnects in VLSI systems, at both physical and architectural levels.

Uri C. Weiser is a visiting Professor at the Electrical Engineering department, Technion IIT and acts as an advisor at numerous startups. He received his bachelor and master degrees in EE from the Technion and a Ph.D. in CS from the University of Utah, Salt Lake City. Uri worked at Intel from 1988–2006. At Intel, Uri initiated the definition of the first Pentium processor, drove the definition of Intel's MMX technology, invented (with A. Peleg) the Trace Cache, he co-managed and established the Intel Microprocessor Design Center at Austin, Texas and later initiated an Advanced Media applications research activity. Uri was appointed Intel Fellow in 1996, in 2002 he became IEEE Fellow and in 2005 ACM Fellow. Prior to his career at Intel, Uri worked for the Israeli Department of Defense as a research and system engineer and later with National Semiconductor Design Center in Israel, where he led the design of the NS32532 microprocessor. Uri was an Associate Editor of *IEEE Micro Magazine* (1992–2004) and was Associate Editor of *Computer Architecture Letters*.

References

[1] L. O. Chua, "Memristor: The missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sept. 1971.

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

[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[4] [Online]. Available: http://www.theinquirer.net/inquirer/news/2234321/sandisk-and-toshiba-dabble-in-reram

[5] [Online]. Available: http://www.micron.com/products/phase-change-memory

[6] [Online]. Available: http://www.everspin.com/PDF/ST-MRAM_Press_Release.pdf

[7] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY logic design procedure," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 2011, pp. 142–147.

[8] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Hybrid CMOS-memristor logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, submitted for publication.

[9] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinalli, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Mederios-Riberio, and R. S. Williams, "Memristor-CMOS hybrid integrated circuits for reconfigurable logic," *Nano Lett.*, vol. 9, no. 10, pp. 3640–3645, Oct. 2009.

[10] Y. V. Pershin and M. Di Ventra, "Practical approach to programmable analog circuits with memristors," *IEEE Trans. Circuits Syst. I. Reg. Papers*, vol. 57, no. 8, pp. 1857–1864, Sept. 2010.

[11] D. Bondurant, B. Engel, and J. Slaughter, "MRAM: The future of non-volatile memory?" *Portable Design*, July 2008.

3.2 Logic Circuits

This section contains the following papers:

- S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY Logic Design Flow," *Proceedings of the IEEE International Conference on Computer Design*, pp.142-147, October 2011.
- S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI)* (in press).
- S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC – Memristor Aided LoGIC," *IEEE Transactions on Circuits and Systems II: Express Briefs* (in review).
- S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic," *Proceedings of the International Cellular Nanoscale Networks and their Applications*, pp. 1-6, August 2012.
- S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic for Hybrid CMOS-Memristor Circuits," *IEEE Transactions on Nanotechnology* (in review).
- Y. Levy, J. Bruk, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaacobi, and S. Kvatinsky, "Logic Operation in Memory Using a Memristive Akers Array," *Microelectronics Journal* (in press).

Memristor-based IMPLY Logic Design Procedure

Shahar Kvatinsky, Avinoam Kolodny,
and Uri C. Weiser

Department of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 32000 ISRAEL
{skva@tx, kolodny@ee, uri.weiser@ee}.technion.ac.il

Eby G. Friedman

Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627 USA
friedman@ece.rochester.edu

***Abstract* — Memristors can be used as logic gates. No design methodology exists, however, for memristor-based combinatorial logic. In this paper, the design and behavior of a memristive-based logic gate – an IMPLY gate - are presented and design issues such as the tradeoff between speed (fast write times) and correct logic behavior are described, as part of an overall design methodology. A memristor model is described for determining the write time and state drift. It is shown that the widely used memristor model - a linear ion drift memristor - is impractical for characterizing an IMPLY logic gate, and a different memristor model is necessary such as a memristor with a current threshold.**

Keywords - memristor; memristive systems; IMPLY; design methodology; logic

I. INTRODUCTION

Memristors are passive elements with varying resistance (also known as a memristance), conceived theoretically in [1]. Changes in the memristance depend upon the history of the device, the total charge which passes through it, or, alternatively, the total flux in the device (the integral over time of the applied voltage at the ports of the device).

In 2008, Hewlett-Packard announced the fabrication of a working memristor [2]. A linear ion drift model was proposed for describing the behavior of this memristor. The memristance of a linear ion drift memristor is

$$M(q) = R_{OFF}\left(1 - \frac{\mu_v R_{ON}}{D^2} q(t)\right), \quad (1)$$

where R_{OFF} and R_{ON} are, respectively, the maximum and minimum resistance of the memristor, μ_v is the average ion mobility, D is the memristor physical thickness, and $q(t)$ is the total charge passing through the memristor. The linear ion drift model is the most commonly used memristor model, although practical memristors exhibit highly non-linear behavior.

Memristors can be used for numerous applications, such as memory [3], neuromorphic systems [4], and analog circuits (e.g., see [5]). One interesting application of memristors is logic, using memristors as building blocks of logic gates. To use memristors in a digital manner, a high memristance is considered as logic *0* and a low memristance is considered as logic *1*. Several approaches for memristor-based logic have been proposed, e.g., [6] and [7], which suggest using memristors as configurable switches as in an FPGA. The logic gates are designed as CMOS gates or as programmable majority logic array (PMLA) based on Goto pairs as logic gates [8].

Another approach is to use memristors as the primary building blocks of a logic gate. Each memristor acts as an input, output, computational logic element, and a latch in different stages of the computing process [9]. In [10], a memristor-based logic gate - the IMPLY gate, is presented. Since this logic function together with FALSE (a function that always yields the value *0* as an output) comprise a computationally complete logic structure, it may potentially provide a basic logic element for a memristor-based circuit. The truth table for *p IMPLY q* is listed in Table 1. Unlike CMOS logic [11], no design methodology exists for memristor-based logic circuits.

In this paper, a design methodology is suggested for memristor-based IMPLY logic gates. A memristor-based IMPLY gate and related limitations are also presented here. The tradeoff between performance and robustness is described as well as the necessity to refresh the logic gate.

This paper is organized as follows. In Section II, the operation of a memristor-based IMPLY gate is described. In section III, the performance and limitations of this logic gate are presented. In section IV, a design example is described, and simulation results of the IMPLY gate are shown. The paper is summarized in section V.

II. MEMRISTOR-BASED IMPLY GATE

The logic function $p \rightarrow q$ (also known as "*p IMPLIES q*," "*material implication*," and "*if p then q*") is described in [10]. The proposed memristor logic is based upon a resistor R_G ($R_{ON} < R_G < R_{OFF}$) connected to two memristors, named P and Q, acting as digital switches. The corresponding initial memristances p and q are the inputs of the gate; while the output of the gate is the final memristance of Q (the result is written into the logic state q). A schematic of an IMPLY gate is shown in Figure 1.

The basic concept is to apply different negative voltages to P and Q, where V_{SET}, the applied voltage on Q, has a higher magnitude than V_{COND}, the applied magnitude on P ($|V_{COND}| < |V_{SET}|$). If $p = 1$ (low resistance), the voltage on the common terminal is approximately V_{COND} and the voltage on

This work was partially supported by Hasso Plattner Institute and by Intel grant "Heterogeneous Computing, the Inevitable Solution: Power Management, Scheduling and ISA" grant no. 864-737-13.

the memristor Q is approximately V_{SET} - V_{COND}, which is sufficiently small to maintain the logic state of q. In the case of $p = 0$ and $q = 0$ (high resistances), the applied voltage on Q is approximately V_{SET} and Q is switched ON ($q = 1$). In the case of $p = 0$ and $q = 1$, the logic state of q is maintained.

A two input NAND, based on a memristor-based IMPLY gate and a FALSE logic gate, is described in [10]. The circuit is comprised of three memristors; the operation of this NAND gate changes the function of each memristor during the computing process. Two memristors act as inputs in the initial stage, one memristor acts as the output in the last stage, and all memristors act together as a computational logic element (as a memristor-based IMPLY gate) during different stages of the computing process. This application requires three computing stages (one FALSE and two IMPLY). A schematic and the sequence of an IMPLY-based NAND are shown in Figure 2.

The execution of any general Boolean function $f: B^n \rightarrow B$ can be implemented with only $n + 3$ memristors [12], where three additional memristors carry out the computation. Only two memristors are required for up to three inputs. Computation of the function is performed in steps. In each step, either FALSE is applied to one memristor, or an IMPLY is applied to two memristors, where the output is written (which is one of the inputs of the computational IMPLY stage). This process requires a long sequence of operations depending upon the number of inputs. This methodology is improved in [13] where only two additional memristors are used rather than three. While [12] and [13] present a general algorithm to compute any Boolean function with a minimal number of memristors, the computational process requires a large number of functional stages, and therefore requires significant computational time.

III. Design Considerations and Performance Analysis of the Memristor-Based IMPLY Gate

A. Analysis fundamentals

The behavior of a memristor-based IMPLY gate is mathematically cumbersome for analysis. There is therefore a need to develop heuristics for designing memristive circuits.

These heuristics can be extended to enable a complete design methodology for memristor-based circuits. A flow diagram of an IMPLY logic gate design methodology is shown in Figure 3.

In this section, design strategies for choosing the proper circuit parameters (R_G, V_{SET}, and V_{COND}) are discussed. The tradeoff between the delay time of the circuit (to maintain the proper write time) and the number of cycles to refresh the memristors (because of state variable drift) is described.

TABLE 1. Truth table of IMPLY function.

Case	*p*	*q*	*p→q*
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

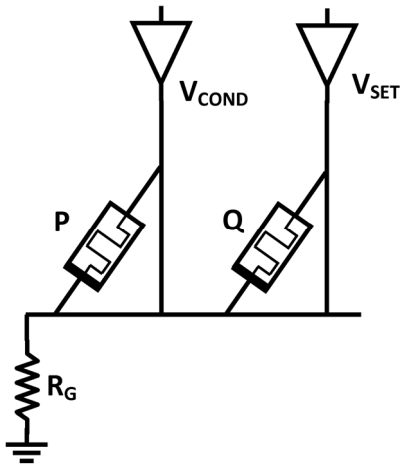
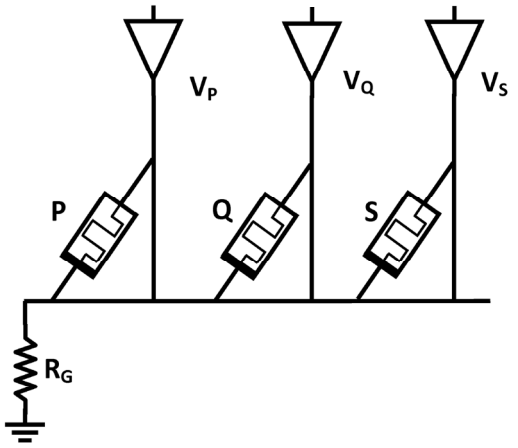


Figure 1. Schematic of a memristor-based IMPLY gate. Two memristors *P* and *Q* are connected to a resistor R_G. The logic state of the memristors *P* and *Q* are, respectively, *p* and *q*.

Step	Voltages		
Step 1: *s=0*			$V_S = V_{CLEAR}$
Step 2: *p→s*	$V_P = V_{COND}$		$V_S = V_{SET}$
Step 3: *q→s*	$V_q = V_{COND}$	$V_S = V_{SET}$	

(a)



(b)

Figure 2. IMPLY NAND logic gate. (a) Logical operation of an IMPLY-based NAND, the logic gate requires three sequential steps, and (b) schematic of IMPLY-based NAND gate.

B. The tradeoff between performance and robustness

V_{SET} and V_{COND}, the applied voltages on P and Q, are fixed. Therefore, for any initial state, the memristor state q tends to drift towards the ON state. For digital operation, the state of q should either stay unchanged or switch fully ON (changing the logic state from logic *0* to logic *1*).

The different input combinations are presented in Table 1. Note that in cases 2 and 4, the initial state of q is logic *1* and the logic gate output q is also logic *1*. The gate operation, therefore, electrically reinforces the logic state of q, and the memristance of Q is reduced.

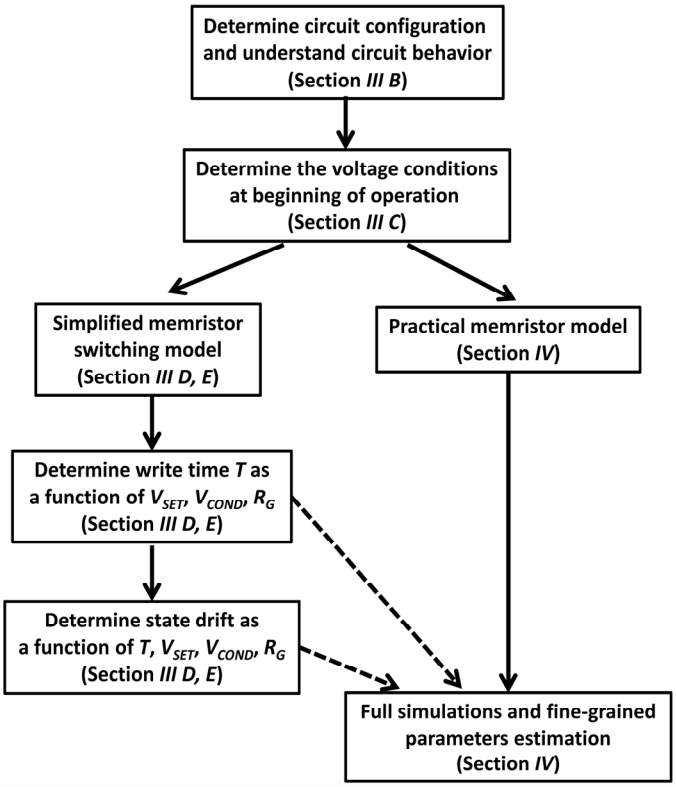


Figure 3. IMPLY logic gate design flow diagram. Each box refers to the relevant section of this paper.

In case 1, the initial state of q is logic *0*; after applying the external voltages, q is switched *ON*. This case determines the time required to apply V_{SET} and V_{COND} until the logic state of q reaches the desired state (above a certain level of conduction to maintain correct logic behavior). This case determines the speed of the circuit in terms of the write time.

In case 3, the initial state of q is logic *0*. This logic state should remain unchanged after applying V_{SET} and V_{COND}, although the voltages tend to change the internal state of q towards the *ON* state of logic *1*. This phenomenon is "state drift." The logic *0* state of q, which is the output of the gate, is electrically "weaker" than the input logic state of q (the memristance of q after applying the voltages is lower than the initial memristance). State drift may require refreshing the state; otherwise, the sensing action may incorrectly switch the logic state of q. State drift depends upon the write time determined for case 1; a long write time increases the state drift phenomenon.

C. Basic principles for parameter determination and design procedure

Although it is difficult to compute the precise value of the applied voltage on Q, it is possible to determine the applied voltage on Q at the beginning of the logic gate activity. The initial applied voltage on Q is different for each case (a different initial memristance for q and p). The initial applied voltages on P and Q are listed in Table 2 under the assumptions that the memristance of logic 1 and logic 0 is, respectively, R_{ON} and R_{OFF}, where $R_{OFF} >> R_{ON}$.

From the initial applied voltages, some necessary conditions for correct logic behavior can be determined. These conditions are not precise, but can provide design constraints. The basic design principle is that the write time of the logic gate is determined from case 1, but the parameters of the circuit should also not exceed a specific state drift in case 3. To determine the circuit parameters, an effective model for the memristors needs to be chosen. The model needs to be sufficiently accurate, while also correctly representing the switching behavior. Inserting the initial applied voltages into the simple memristor switching model can provide an approximate estimate of the circuit parameters.

D. Write time and state drift for a binary memristance

A useful and simple switching model is the binary memristance model. Assume only two allowed memristances, R_{ON} and R_{OFF}. A total charge Q' must flow through the memristor to cause the memristance R_{OFF} to switch to memristance R_{ON}. Under these assumptions and by solving both the switching behavior in case 1 and the write time T as a function of Q', the circuit parameter T is

$$T = \left[\frac{R_{OFF}^{\ 2} + 2R_{OFF}R_G}{R_{OFF}V_{SET} + R_G\left[V_{SET} - V_{COND}\right]} \right] \cdot Q'. \quad (2)$$

The write time for different circuit parameters and a varying V_{SET} is shown in Figure 4. Note that the logic gate is faster with higher applied voltages, or smaller R_{OFF}.

Under this model, it is possible to limit the state drift (case 3) for a fixed drift. The state drift is

$$q_q(T) \approx \left[V_{SET} - \frac{R_G}{R_{ON} + R_G} V_{COND} \right] \cdot \left[\frac{R_{OFF} + 2R_G}{R_{OFF}V_{SET} + R_G\left[V_{SET} - V_{COND}\right]} \right] \cdot Q', \quad (3)$$

where $q_q(T)$ is the total charge flowing through memristor Q after time T in case 3. To limit the state drift to a value of $Q'/4$, after four times, the logic gate is applied as in case 3, and the state drift changes the memristive logic state. This phenomenon requires a refresh every three times the gate is used, since the logic state changes during the fourth time. The allowed value of V_{SET} for several circuit parameters is shown in Figure 5. Note that the state drift is more significant with a higher applied voltage, or with smaller R_{OFF}. Combining Figures 4 and 5, the tradeoff between the speed and robustness of a memristive logic gate is shown in Figure 6.

E. R_G for a fixed threshold model

Another simple memristor model assumes non-linear behavior with a fixed threshold voltage V_{ON}. For an applied voltage below V_{ON}, the memristance is unchanged. To produce correct logical behavior, the initial applied voltage on Q must be above the threshold voltage in case 1 and below the threshold voltage in case 3. Adding this assumption to the initial applied voltage (see Table 2) leads to the following two conditions on the circuit parameters,

TABLE 2. APPLIED LOGIC GATE VOLTAGES V_Q AND V_P, RESPECTIVELY, ON MEMRISTORS P AND Q AT $t = 0$, UNDER THE ASSUMPTIONS THAT THE MEMRISTANCE OF LOGIC *1* AND LOGIC *0* IS, RESPECTIVELY, R_{ON} AND R_{OFF}, WHERE $R_{OFF} >> R_{ON}$.

Case	$V_Q(t=0)$	$V_P(t=0)$
1	$\frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_G}{R_{OFF}+2R_G}\cdot V_{COND}$	$-\left[\frac{R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{COND}\right]$
2	$V_{SET}\cdot\frac{R_{ON}}{R_{OFF}}\cdot\frac{R_{OFF}+R_G}{R_{ON}+R_G} \approx V_{SET}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+R_G} - V_{COND}\right]$
3	$V_{SET} - V_{COND}\cdot\frac{R_G}{R_{ON}+R_G}$	V_{COND}
4	$V_{SET}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_G}{R_{ON}+2R_G}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G}\right]$

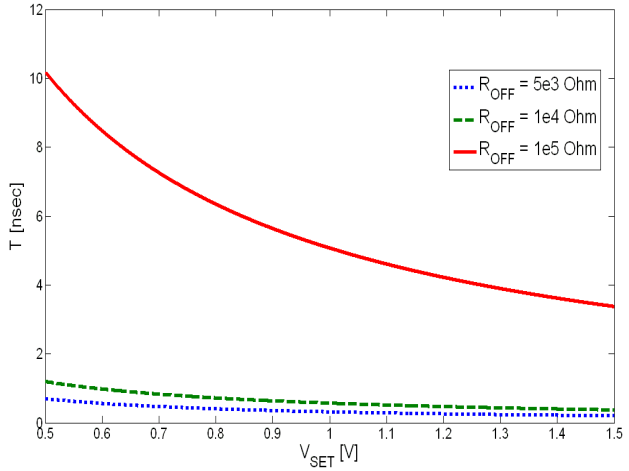


Figure 4. Write time *T* in case 1 for three values of R_{OFF} (5 kΩ, 10 kΩ, and 100 kΩ) under the assumptions of a binary resistance model and Q' = $5\cdot10^{-14}$ C.

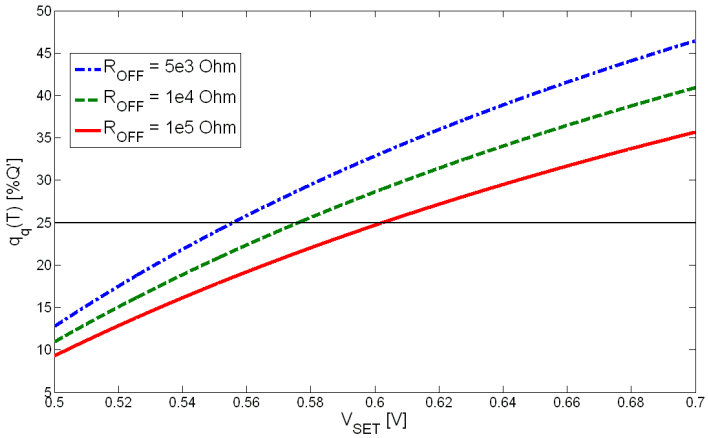


Figure 5. Allowed values of V_{SET} for limited state drift in case 3 of *Q'/4*. V_{SET} is allowed if $q_q(T)$ is smaller than *Q'/4* (the horizontal line in the figure).

$$R_{ON}\cdot\frac{V_{SET}-V_{ON}}{V_{ON}-[V_{SET}-V_{COND}]} < R_G < R_{OFF}\cdot\frac{V_{SET}-V_{ON}}{2V_{ON}-[V_{SET}-V_{COND}]}, \quad (4)$$

$$\frac{V_{SET}}{V_{COND}} < \frac{R_{OFF}}{R_{ON}}. \quad (5)$$

The allowed value for R_G for several circuit parameters and varying V_{SET} are shown in Figure 7.

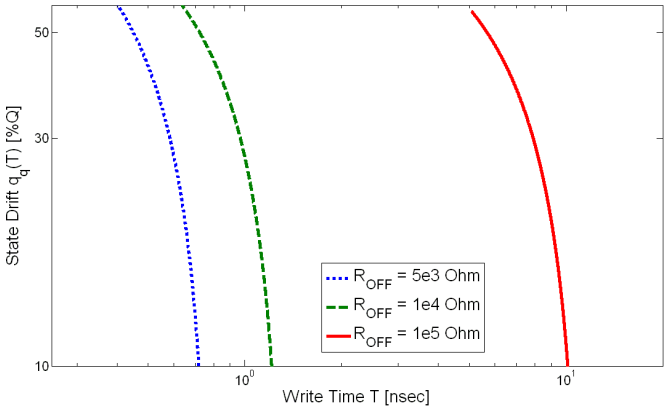


Figure 6. Tradeoff between the logic gate speed (write time) and robustness (the state drift in case 3 for memristor *Q*), for three values of R_{OFF} (5 kΩ, 10 kΩ, and 100 kΩ) under the assumptions of a binary resistance model and Q' = $5\cdot10^{-14}$ C.

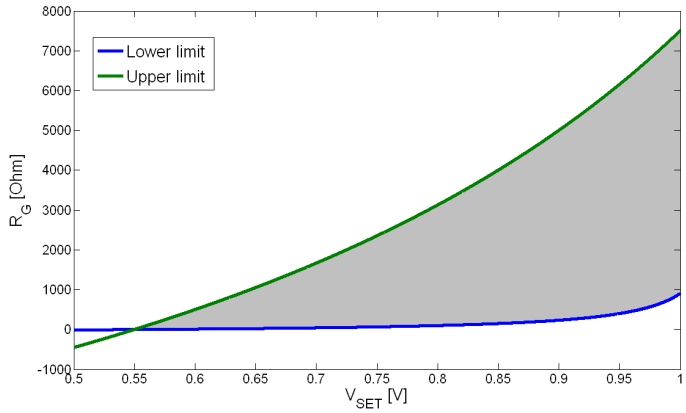


Figure 7. Allowed value of R_G depends on V_{SET}. The upper line is the upper bound for allowed R_G and the lower line is the lower allowed bound for R_G. Under the assumption of a threshold voltage V_{ON} = 0.55 V, V_{COND} = 0.5 V, R_{ON} = 100 Ω, and R_{OFF} = 10 kΩ.

IV. DESIGN EXAMPLE

As a specific example of applying the flow chart of Figure 3, assume the requirements for a circuit are a maximum write time of 0.5 μsec (note that the write time is normalized. A practical memristor write time is significantly faster [14]) and the maximum state drift is $0.025R_{OFF}$ (2.5% of the state drift as compared to full switching).

Assume a memristor with R_{ON} and R_{OFF}, respectively, of 1 kΩ and 100 kΩ. Set one circuit parameter V_{COND} to 0.5 V. The behavior of an ideal IMPLY logic gate (zero write time, no state drift) is shown in Figures 8 and 9. Practical logic gates, however, have non-zero write time and state drift. From Figures 4 and 5, note that as V_{SET} rises, the logic gate write time T decreases and the gate response is faster; however, the state drift phenomenon is more significant. From (5),

$$0.5V < V_{SET} < 50V. \quad (6)$$

This expression only produces a lower bound on V_{SET}, since the upper bounds are significantly higher than practical on-chip supply voltages. For a current-controlled memristor, it is unrealistic to determine an exact equivalent voltage threshold

(which depends on the transient memristance of the device). A good approximation for an equivalent voltage threshold is

$$V_{ON} = i_{ON} \cdot R_{OFF}, \quad (7)$$

where V_{ON} is the voltage threshold, and i_{ON} is the current threshold. For a memristor with a current threshold of 7 μA, the equivalent voltage threshold is 0.7 volts. From (4), R_G is

$$1.5\,k\Omega < R_G < 33.3\,k\Omega. \quad (8)$$

The widely used linear ion drift memristor model [15] is incompatible with IMPLY logic gates. In this model, the memristance changes linearly for any applied voltage; the state drift phenomenon is therefore significant, as shown in Figures 10 and 11. Hence, a different memristor model with a current threshold is preferable [16]. With this model, the exact circuit parameters are selected. The chosen circuit parameters are R_{ON} = 1 kΩ, R_{OFF} = 100 kΩ, V_{COND} = 0.5 V, V_{SET} = 1 V, and R_G = 5 kΩ. SPICE simulation results for these parameters are shown in Figures 12 and 13. The write time and state drift for several circuit parameters are listed in Table 3. An increase in the resistance of R_G or decrease in the voltage level of V_{SET} delays the gate, but lowers the state drift (and vice versa).

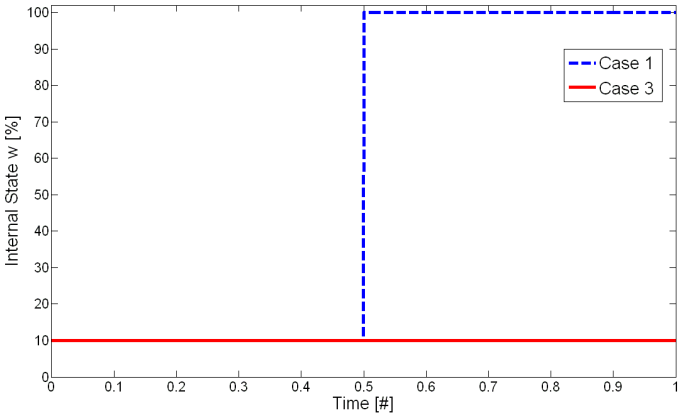


Figure 8. State drift of an ideal IMPLY logic gate. While the logic state in case 1 changes to a zero write time, the drift for case 3 is zero.

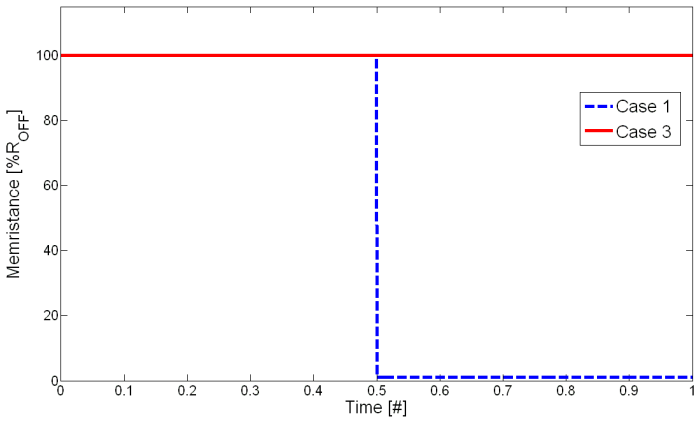


Figure 9. Memristance of an ideal IMPLY logic gate. While the memristance in case 1 decreases to R_{ON} within a zero write time, the memristance in case 3 does not change.

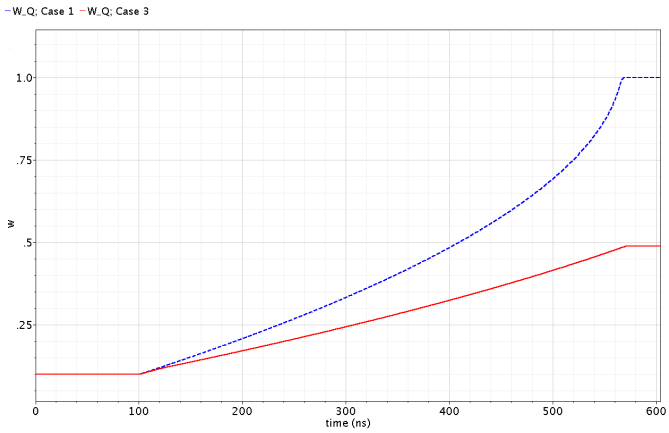


Figure 10. State variable w of q when applying IMPLY logic gate for cases 1 (dashed line) and 3 (solid line) for a memristor with linear ion drift. T is 468.1 nsec. The state drift for case 3 is 48.9%, which makes this model impractical for an IMPLY logic gate.

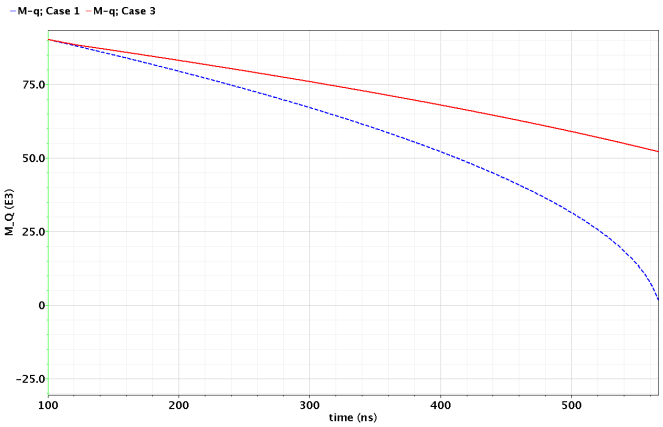


Figure 11. The memristance of q when applying an IMPLY logic gate for cases 1 (dashed line) and 3 (solid line) for a memristor with linear ion drift.

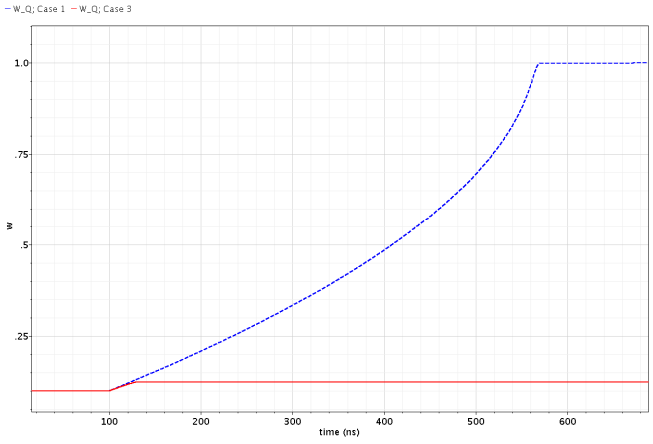


Figure 12. State variable w of q when applying an IMPLY logic gate for cases 1 (dashed line) and 3 (solid line) for a memristor with a threshold model (current threshold is 7 µA). T is 470.3 nsec. The state drift for case 3 is 2.44%.

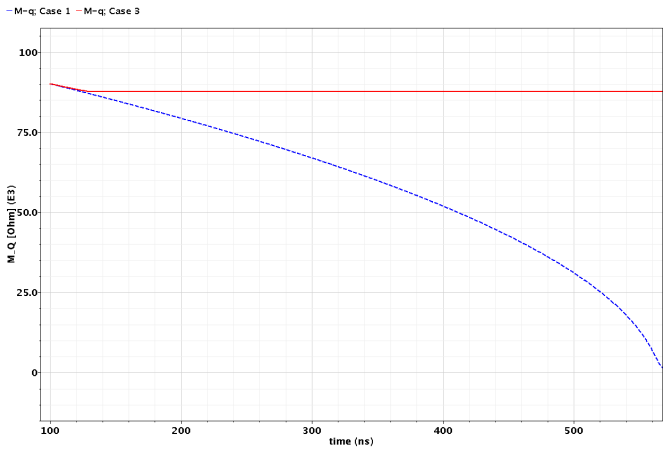


Figure 13. Memristance of *q* when applying an IMPLY logic gate for cases 1 (dashed line) and 3 (solid line) for a memristor with threshold model (current threshold is 7 µA).

TABLE 3. WRITE TIME AND STATE DRIFT FOR DIFFERENT VALUES OF V_{SET} AND R_G. ALL VALUES SATISFY (6) AND (8). V_{COND} IS SET TO 0.5 V.

V_{SET} [V]	R_G [kΩ]	T [µsec]	State Drift [% R_{OFF}]
1	5	0.47	2.44
0.8	5	0.592	~ 0
1.5	5	0.31	6
1	3.5	0.453	2.53
1	15	0.579	2.15

V. CONCLUSIONS

The logic design of a memristor-based IMPLY logic gate is presented. Investigating and characterizing the behavior of a memristor and IMPLY logic gate reveals several design limitations and considerations. The IMPLY logic gate trades off performance (write time) with robustness (internal state drift). This tradeoff requires the circuit to be occasionally refreshed.

Several heuristics for designing IMPLY logic gates with memristors are proposed and organized into a design procedure. This design procedure considers the influences and tradeoffs among the different input cases, initial conditions, and circuit parameters of the memristor.

A design example based on the proposed design procedure is presented and compared with simulation. It is shown that the widely used linear ion drift model is incompatible with the IMPLY logic gate, since under this model, the state drift phenomenon is excessively high. To accurately characterize the IMPLY logic gate operation, a highly non-linear memristor model needs to be used; or alternatively, a device with a threshold. The proposed design procedure is the first step in the development of a general design methodology for logic gates based on memristors.

REFERENCES

[1] L. O. Chua, "Memristor – the Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[2] D. B. Strukov, G. S.Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, Vol. 453, pp. 80-83, May 2008.

[3] Y. Ho, G. M. Huang, P. Li, "Nonvolatile Memristor Memory: Device Characteristics and Design Implications," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 485-490, November 2009.

[4] A. Afifi, A. Ayatollahi, and F. Raissi, "Implementation of Biologically Plausible Spiking Neural Network Models on the Memristor Crossbar-based CMOS/Nano Circuits," *Proceedings of the European Conference on Circuit Theory and Design*, pp. 563- 566, August 2009.

[5] Y. V. Pershin and M. Di Ventra, "Practical Approach to Programmable Analog Circuits with Memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 8, pp. 1857-1864, August 2010.

[6] D. B. Strukov and K. K. Likharev, "CMOL FPGA: a Reconfigurable Architecture for Hybrid Digital Circuits with Two-Terminal Nanodevices," *Nanotechnology*, Vol. 16, No. 6, pp. 888-900, June 2005.

[7] G. S. Snider and R. S. Williams, "Nano/CMOS Architectures Using a Field-Programmable Nanowire Interconnect," *Nanotechnology*, Vol. 18, No. 3, 035204, January 2007.

[8] G. S. Rose and M. R. Stan, "A Programmable Majority Logic Array Using Molecular Scale Electronics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 54, No. 11, pp. 2380-2390, November 2007.

[9] G. Snider, "Computing with Hysteretic Resistor Crossbars," *Applied Physics A: Materials Science and Processing*, Vol. 80, No. 6, pp. 1165-1172, March 2005.

[10] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable 'Stateful' Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873-876, April 2010.

[11] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison Wesley, 2010.

[12] E. Lehtonen and M. Laiho, "Stateful Implication Logic with Memristors," *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 33-36, July 2009.

[13] E. Lehtonen, J. H. Poikonen, and M. Laiho, "Two Memristors Suffice to Compute All Boolean Functions," *Electronics Letters*, Vol. 46, No. 3, pp. 239-240, February 2010.

[14] K. Eshraghian, K. R. Cho, O. Kavehei, S. K. Kang, D. Abbot, and S. M. S. Kang, "Memristor MOS Content Addressable Memory (MCAM): Hybrid Architecture for Future High Performance Search Engines," *IEEE Transactions on Very Large Scale Integrated Systems*, in press.

[15] Z. Biolek, D. Biolek, and V. Biolkova, "Spice Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, Vol. 18, No. 2, Part 2, pp. 210-214, June 2009.

[16] S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," unpublished.

Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies

Shahar Kvatinsky, *Student Member, IEEE*, Guy Satat, Nimrod Wald, Eby G. Friedman, *Fellow, IEEE*, Avinoam Kolodny, *Senior Member, IEEE*, and Uri C. Weiser, *Fellow, IEEE*

***Abstract*— Memristors are novel devices, useful as memory at all hierarchies. These devices can also behave as logic circuits. In this paper, the IMPLY logic gate, a memristor-based logic circuit, is described. In this memristive logic family, each memristor is used as an input, output, computational logic element, and latch in different stages of the computing process. The logical state is determined by the resistance of the memristor. This logic family can be integrated within a memristor-based crossbar, commonly used for memory. In this paper, a methodology for designing this logic family is proposed. The design methodology is based on a general design flow, suitable for all deterministic memristive logic families, and includes some additional design constraints to support the IMPLY logic family. An IMPLY 8-bit full adder based on this design methodology is presented as a case study.**

***Index Terms*— Design methodology, IMPLY, logic, memristive systems, memristor, Von Neumann architecture.**

I. Introduction

MEMRISTORS [1] and memristive devices [2] are novel structures, useful in many applications. These devices are basically resistors with varying resistance, which depends on the history of the device. It can be used for memory, where the data is stored as a resistance. While memory is the common application for memristive devices, additional applications can also use memristive devices as functional blocks, such as analog circuits, neuromorphic systems, and logic circuits. Although the definition of memristive devices is broader than the definition of memristors, it is common to use the term memristor for all memristive devices [10], [11]. In this paper, for simplicity, the terms memristor and memristive device are used interchangeably.

The use of memristors to perform logical operations has been proposed in several different ways. In some logic families, memristors are integrated with CMOS structures to perform the logical operation, while the logical values are represented by voltage levels. In [3], memristors are used as a reconfigurable switch. In [4], a hybrid memristor-CMOS logic family is proposed, memristor ratioed logic (MRL). In MRL, the memristors act as computational elements, performing OR and AND Boolean functions, while the CMOS transistors perform logical inversion and amplification of the logical voltage signals. A similar approach is proposed in [5].

Another approach for logic with memristors is to treat resistance as the logical state, where the high and low resistance are considered, respectively, as logical zero and one. For this approach, the memristors are the primary building blocks of the logic gate. Each memristor acts as an input, output, computational logic element, and latch in different stages of the computing process [6]. This approach is suitable for crossbar array architectures and can therefore be integrated within a standard memristor-based crossbar, commonly used for memory. This approach is appealing since it provides an opportunity to explore advanced computer architectures different from the classical von Neumann architecture. In these architectures, the memory can perform logical operations on the same devices that store data, i.e., performing computation inside the memory. This paper focuses on this approach.

Material implication (IMPLY logic gate) [7] is one example of a basic logical element using this approach, combining state memory and a Boolean operator. Additional logic families, which extends the IMPLY logic gate by using certain variations of a regular memristor-based crossbar, have also been proposed [8], [9] and are not considered in this paper. A specific modification of the crossbar structure is, however, presented in this paper to enhance the performance of the logic gate.

In this paper, the IMPLY logic gate is described in Section III, and a memristor-based crossbar in Section IV. A design methodology for the IMPLY logic gate is proposed in Section V. This design methodology consists of a design flow appropriate for all memristor-based logic families, as well as the IMPLY logic family. This design methodology is demonstrated by a case study of an 8-bit IMPLY full adder in Section VI. Logic inside a memristor-based memory is discussed in Section VII. This paper is concluded in Section VIII.

II. Memristors

Memristors were conceived in 1971 by Chua [1] based on fundamental principles of symmetry. Chua proposed a fourth fundamental electronic component in addition to the three already well-known fundamental electronic components: the resistor, capacitor, and inductor. The memristor has varying

Manuscript received February 23, 2013; revised June 1, 2013 and August 26, 2013; accepted September 8, 2013. This work was supported in part by the Hasso Plattner Institute, in part by the Advanced Circuit Research Center at Technion, and in part by the Intel Collaborative Research Institute for Computational Intelligence.

S. Kvatinsky, G. Satat, N. Wald, A. Kolodny, and U. C. Weiser are with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel (e-mail: skva@tx.technion.ac.il; guysatat@hotmail.com; nimrodwald@gmail.com; kolodny@ee.technion.ac.il).

E. G. Friedman is with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627 USA (e-mail: friedman@ece.rochester.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2282132

Fig. 1. Memristive device symbol. The thick black line on the left side of the device represents the polarity of the device. If the current flows into the device, the resistance of the device decreases. If the current flows out of the device, the resistance increases.

resistance (also named memristance). Changes in the memristance depend upon the history of the device (e.g., the memristance may depend on the total charge passing through the device, or alternatively, on the integral over time of the applied voltage across the ports of the device).

The theory of memristors was extended to memristive devices in 1976 [2]. Formally, a current-controlled time-invariant memristive system is represented by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, i) \tag{1}$$

$$v(t) = R(x, i) \cdot i(t) \tag{2}$$

where x is an internal state variable, i(t) is the memristive device current, $v(t)$ is the voltage of the memristive device, $R(x, i)$ is the memristance, and t is time. The symbol of a memristor is illustrated in Fig. 1. Note that the polarity of the symbol defines the sign (positive or negative) of the current.

Since Hewlett–Packard announced the fabrication of a working memristor in 2008 [12], there has been increasing interest in memristors and memristive systems. New devices exhibiting memristive behavior have been announced [13], [14], and existing devices such as spin-transfer torque magnetoresistive random access memory (STT-MRAM) have been redescribed in terms of memristive systems [15]. Actually, most emerging memory technologies obey (1) and (2) and can therefore be described as memristive devices or memristors [11].

Several memristor models have been proposed to describe the behavior of physical memristors [16]–[23]. These models are deterministic and do not consider stochastic switching [40], [41]. In this paper, the threshold adaptive memristor (TEAM) model [23] is used. In the TEAM model, memristors have an adaptive nonlinearity and a current threshold. For this model, (1) becomes

$$\frac{dx(t)}{dt} = \begin{cases} k_{\mathrm{OFF}} \cdot \left(\frac{i(t)}{i_{\mathrm{OFF}}} - 1\right)^{\alpha_{\mathrm{OFF}}} \cdot f_{\mathrm{OFF}}(x), & 0 < i_{\mathrm{OFF}} < i \quad \text{3(a)} \\ 0, & i_{\mathrm{ON}} < i < i_{\mathrm{OFF}} \quad \text{3(b)} \\ k_{\mathrm{ON}} \cdot \left(\frac{i(t)}{i_{\mathrm{ON}}} - 1\right)^{\alpha_{\mathrm{ON}}} \cdot f_{\mathrm{ON}}(x), & i < i_{\mathrm{ON}} < 0 \quad \text{3(c)} \end{cases}$$

where k_{OFF} and k_{ON} are fitting parameters, α_{ON} and α_{OFF} are the adaptive nonlinearity parameters, i_{OFF} and i_{ON} are the current threshold parameters, and $f_{\mathrm{ON}}(x)$ and $f_{\mathrm{OFF}}(x)$ are window functions. An I–V curve for the TEAM model is shown in Fig. 2 for memristors where (2) is

$$v(t) = \left[R_{\mathrm{ON}} + \frac{R_{\mathrm{OFF}} - R_{\mathrm{ON}}}{x_{\mathrm{OFF}} - x_{\mathrm{ON}}}(x - x_{\mathrm{ON}})\right] \cdot i(t) \tag{4}$$

where R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, and x_{ON} and x_{OFF} are, respectively, the minimum and maximum allowed value of the internal state variable x.

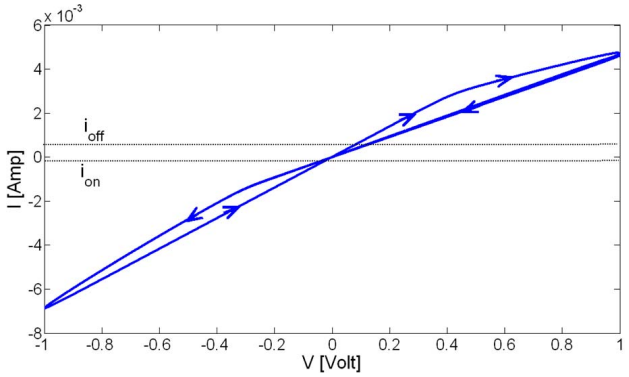


Fig. 2. I–V curve of a memristor based on the TEAM model driven with a sinusoidal input of 1 volt, where $R_{\mathrm{ON}} = 50\Omega$, $R_{\mathrm{OFF}} = 1$ kΩ, $k_{\mathrm{OFF}} = 1.46e^{-9}$ nm/s, $\alpha_{\mathrm{OFF}} = 10$, $i_{\mathrm{OFF}} = 115$ μA, $k_{\mathrm{ON}} = -4.68e^{-13}$ nm/s, $\alpha_{\mathrm{ON}} = 10$, $i_{\mathrm{ON}} = 8.9$ μA, $x_{\mathrm{ON}} = 1.2$ nm, and $x_{\mathrm{OFF}} = 1.8$ nm.

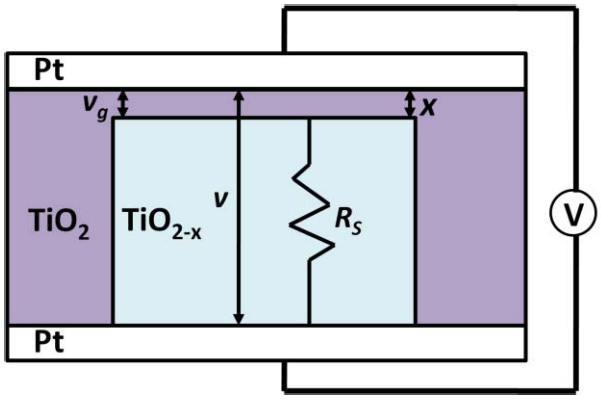


Fig. 3. Schematic of the physical model proposed in [20] for a TiO_2 memristor.

Memristors are nonvolatile and compatible with standard CMOS technologies [24]. These devices are fabricated in the metal layers of an integrated circuit, where the memristive effects occur in the oxide between the metal layers (e.g., in TiO_2 and TaO_x) [25] or within the metal layers (e.g., in STT-MRAM). The physical model of a TiO_2 memristor, proposed in [20], is shown in Fig. 3. The size of a typical memristor is relatively small, since the fabrication process is similar to processing the cross-layer via between metal layers. Memristors therefore exhibit high density and good scalability. The read and write time for these devices can be as fast as 120 picoseconds [25]. Currently, except for STT-MRAM, memristors suffer from endurance limitations, where the number of allowed writes per cell is approximately 10^{10} [26]. It is believed, however, that this limit will increase to at least 10^{15} [27]. Memristors may therefore solve many significant problems in the semiconductor industry, providing nonvolatile, dense, fast, and power-efficient memory.

III. IMPLY Logic Gate

The logic function p→q or p IMPLY q (also known as p IMPLIES q, material implication, and if p then q) is described in [7] and a truth table is listed in Table I. The IMPLY logic

TABLE I
TRUTH TABLE OF IMPLY FUNCTION

Case	*p*	*q*	*p→q*
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

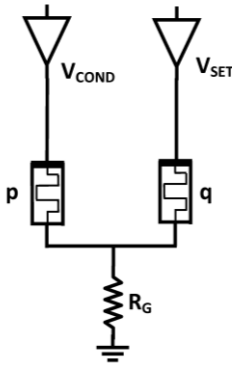


Fig. 4. IMPLY logic gate. The initial state of memristors p and q is the input of the logic gate and the output is the final state of the memristor q after applying the voltages V_{SET} and V_{COND}. A load resistor R_G is connected to both memristors.

function together with FALSE (a function that always yields the value zero as an output) comprises a computationally complete logic structure. Since the IMPLY function can be integrated within a memristor-based crossbar, IMPLY logic provides a basic logic element for a memristor-based circuit.

A. Basic Logic Gate Operation

The proposed memristor-based IMPLY logic gate uses a resistor R_G ($R_{\text{ON}} < R_G < R_{\text{OFF}}$) connected to two memristors, named P and Q, acting as digital switches. The corresponding initial memristances p and q are the inputs of the gate; while the output of the gate is the final memristance of Q (the result is written into the logic state q). Note that the memristance of both memristors changes during operation, i.e., the computation is destructive to both inputs. A schematic of an IMPLY gate is shown in Fig. 4.

The basic concept is to apply two different voltages to P and Q, where V_{SET}, the applied voltage on Q, has a higher magnitude than V_{COND}, the applied magnitude on P ($|V_{\text{COND}}| < |V_{\text{SET}}|$). If $p = 1$ (low resistance), the voltage on the common terminal is approximately V_{COND} and the voltage on the memristor Q is approximately $V_{\text{SET}} - V_{\text{COND}}$, which is sufficiently small to maintain the logic state of q. In the case of $p = 0$ and $q = 0$ (high resistances), the applied voltage on Q is approximately V_{SET} and Q is switched ON ($q = 1$). In the case of $p = 0$ and $q = 1$, the logic state of q is maintained. The memristance of an ideal IMPLY logic gate (zero delay time) for input cases 1 and 3 is shown in Fig. 5.

B. Analyzing the Behavior of a Logic Gate

V_{SET} and V_{COND}, the applied voltages on P and Q, are fixed. For any initial state, the memristor state q tends to drift

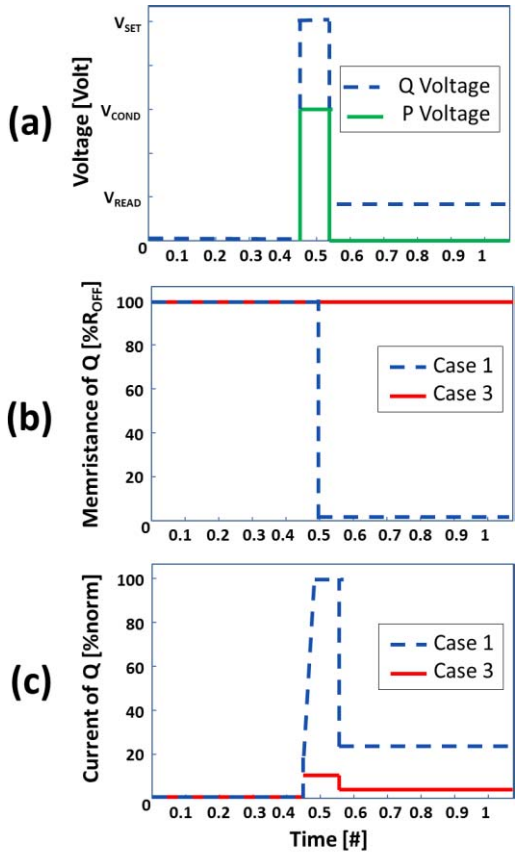


Fig. 5. Behavior of an ideal IMPLY logic gate. (a) Applied voltages on both memristors P and Q. (b) Memristance of Q for cases 1 and 3. While the memristance in case 1 decreases to R_{ON} within a zero write time, the memristance in case 3 does not change. (c) Current of memristor Q. The current in case 1 is sufficiently high to decrease the resistance of Q.

toward the ON state. For digital operation, the state of q should either stay unchanged or switch fully ON (changing the logic state from logical zero to logical one).

The different input combinations are listed in Table I. Due to the polarity of the memristors and the applied voltages, the memristance of memristor Q can only be reduced. Note that in cases 2 and 4, the initial logic state of q is logical one and the logic gate output q is also logical one. The gate operation, therefore, electrically reinforces the logic state of q since the memristance of Q is reduced.

In case 1, the initial state of q is logical zero; after applying the external voltages, q is switched ON. This case determines the time required to apply V_{SET} and V_{COND} until the logic state of q reaches the desired state (above a certain level of conduction that maintains correct logical behavior). This case determines the write time of the circuit (the delay time of the logic gate).

In case 3, the initial state of q is logical zero. This logic state should remain unchanged after applying V_{SET} and V_{COND}, although the voltages tend to change the internal state of q toward the ON state of logical one. This phenomenon is state drift. The logical zero state of q, which is the output of the gate, is electrically weaker than the input logical state of q (the memristance of Q after applying the voltages is lower than the initial memristance). State drift may require

refreshing the state; otherwise, repeated or prolonged sensing action may incorrectly switch the logic state of q. Note that the state drift phenomenon is a deterministic phenomenon. Stochastic switching [40], [41] can change the logical state of the memristors, and is not considered in this paper.

C. Speed–Robustness Tradeoff

The permissible value of the time required to apply V_{COND} and V_{SET} is determined from case 1. This write time is the delay time of the logic gate and determines the performance of the logic gate. Since the initial logical state of the memristors is unknown during operation (no preliminary read operation is applied), the voltages are applied at the same time for all input cases.

The state drift is determined from case 3, which depends upon the write time determined for case 1. Furthermore, any improvement in the performance due to changes in the applied voltage increases the state drift and degrades the robustness of the logic gate [28].

D. Extended Logic Functions Based on IMPLY

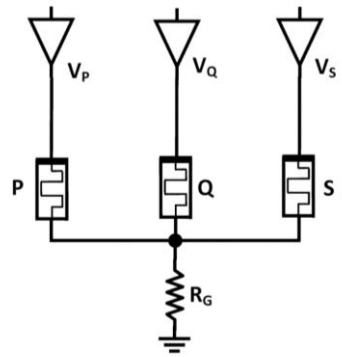
Any general Boolean function $f:B^n \rightarrow B$ can be implemented with only $n+3$ memristors [29], where three additional memristors carry out the computation. Only two memristors are required for up to three inputs. Computation of the function is performed in steps. In each step, either FALSE is applied to one memristor, or an IMPLY is applied to two memristors, where the output is written to a memristor (which is one of the inputs of the computational IMPLY stage). This process requires a long sequence of operations depending upon the number of inputs. This methodology has been improved in [30], where only two additional memristors are used rather than three. While a general algorithm to compute any Boolean function with a minimal number of memristors has been developed [29], [30], the computational process requires a large number of functional stages, and therefore requires significant computational time.

The schematic and sequence of a two input NAND, based on a memristor-based IMPLY gate and a FALSE logic gate, are shown in Fig. 6. This NAND gate is designed to minimize the computational time and number of memristors and is comprised of three memristors. The operation of this NAND logic gate changes the function of each memristor during the computing process. Two memristors act as inputs in the initial stage, one memristor acts as the output in the last stage, and all memristors act together as a computational logic element (as a memristor-based IMPLY gate) during different stages of the computing process. This application requires three computing stages (one FALSE and two IMPLY).

The IMPLY logic gate can also be extended to a multiple input NOR logic gate [31]. In this extension, as illustrated in Fig. 7(a), k input memristors $P_1, P_2, \ldots, P_k$, and a separate output memristor Q are assumed. The operation of this NOR gate requires two computational stages, the first stage initializes Q to logical zero ($q = 0$) and the second stage applies V_{SET} and V_{COND} in a manner similar to regular IMPLY. The extended NOR suffers from low fan-in since R_G needs to be scaled to all possible number of inputs. To solve this issue, a different structure has been proposed where a load resistor R_G is connected to every memristor and the load resistance varies, as shown in Fig. 7(b).

Step	Voltages	
Step 1: s=0		$V_S = V_{CLEAR}$
Step 2: $p \rightarrow s$	$V_P = V_{COND}$	$V_S = V_{SET}$
Step 3: $q \rightarrow s$	$V_q = V_{COND}$	$V_S = V_{SET}$

(a)



(b)

Fig. 6. IMPLY NAND, (a) The logic gate requires three sequential steps. (b) Schematic of IMPLY-based NAND gate.

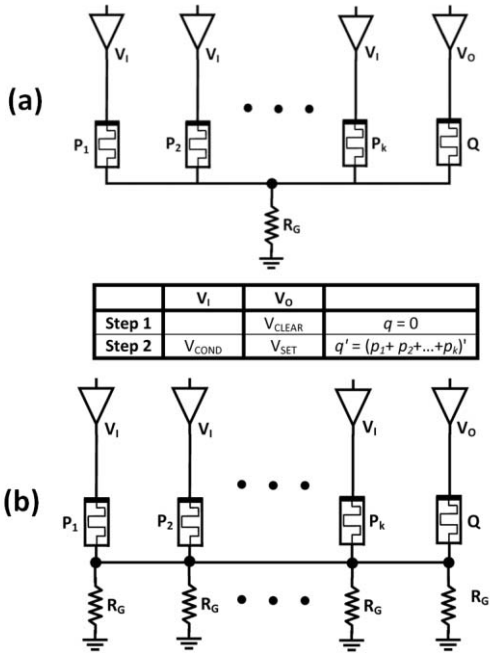


Fig. 7. Extension to IMPLY, a k-input NOR. (a) Schematic based on execution of multiple implications in a single step and (b) improved fan-in structure, where the load resistors are dedicated to the participating logic devices.

IV. IMPLY Inside a Memristor-Based Crossbar

The IMPLY logic gate cannot be easily integrated with standard CMOS logic since both circuit structures are significantly different. In the IMPLY logic family, a resistance, rather than a voltage, represents the logical state. Furthermore, to operate the logic gate, a sequence of specific voltages is

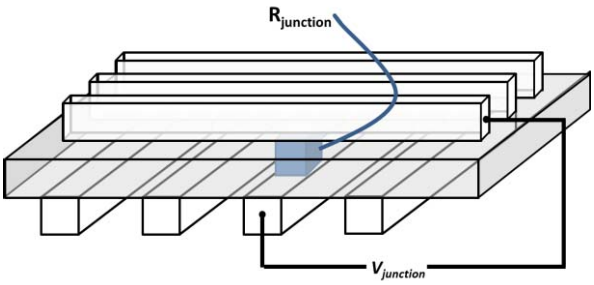


Fig. 8. Basic structure of a memristor-based crossbar. Each junction of the parallel lines is a memory cell with varying resistance R_{junction}.

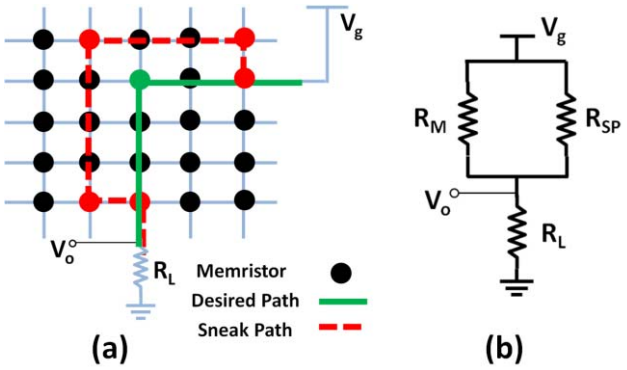


Fig. 9. Sneak path in a memristive crossbar. (a) Example sneak path. Every node in the grid is a memristor. The desired path is marked by a solid line and a sneak path is marked by a dashed line. (b) Equivalent circuit. All sneak paths have an equivalent resistance R_{SP} connected in parallel to the resistance of the memristor R_M.

applied to the memristors. The IMPLY logic gate therefore requires several computational stages (usually a different computational stage is executed during each clock cycle), and a separate mechanism to read the result of the computation and control the voltages. To integrate the IMPLY logic gate with standard voltage-based CMOS logic, a conversion mechanism is required. This mechanism includes a sense amplifier as well as additional components. The additional circuitry reduces the efficiency of integrating CMOS with a memristor-based IMPLY logic gate.

Alternatively, the IMPLY logic gate can be integrated inside a memristor-based crossbar array, commonly used for memory, where the input and output are values stored in the memory cells. This integration reduces power and provides an opportunity for novel non-von Neumann architectures. In this section, the basic structure of a memristor-based crossbar is presented, and a version of the IMPLY logic gate is illustrated.

A. Memristor-Based Crossbar

The basic structure of a memristor-based crossbar consists of two sets of parallel conductive (metal) lines. The conductive lines are perpendicular and behave as top and bottom electrodes to the memristive material, located between the lines [33]. The basic structure of a memristor-based crossbar is shown in Fig. 8. The write operation to a cell within the crossbar is achieved by applying a specific voltage to the junction, where a voltage is applied to both lines. For example, to write a logical one (low resistance), a positive voltage is applied to the column line and ground is connected to the row line (a positive voltage is applied to the memristor). To write a logical zero (high resistance), the column line is connected to ground and a positive voltage is connected to the row line (a negative voltage is applied to the memristor). These voltages are sometimes called V_{SET} (positive voltage to write a logical one, not necessarily the same voltage as in IMPLY) and V_{RESET} (negative voltage to write a logical zero). Since memristors are nonvolatile, the data does not change when no voltage is applied to the lines. The crossbar structure allows the density of the memory to be relatively high, since CMOS transistors are not used for each memory cell, but rather only to select the line. This memory structure is more than 20 times denser than DRAM [34].

The read operation of the crossbar is achieved by applying a relatively low voltage (e.g., lower than V_{SET}) to a junction and measuring the current. From Ohm's law, the resistance of the memristor is determined from this measured current. The current measurement is usually achieved by converting the current into a voltage through a voltage divider with a known resistance R_{pu}. The sensed voltage v_s is compared to a known voltage.

An undesired phenomenon in crossbars is sneak paths [35]–[38], which are undesired paths for the current flow. When a voltage is applied to a junction in the crossbar, current also flows through paths different than the desired path. These paths cross more than one memristor and add a resistance in parallel to the resistance of the memristor in the junction being read. An illustration of the sneak path phenomenon is shown in Fig. 9. This parallel resistance depends upon the stored data in the memristors in the undesired paths and changes the sensed voltage v_s from a simple voltage divider between R_{pu} and the resistance of the memristor to a voltage divider between R_{pu} and the total resistance of all memristors in all paths. A practical sensing operation should therefore consider all possible sneak paths. A schematic of a crossbar, including the read and write mechanisms, is depicted in Fig. 10. Several approaches exist to eliminate or reduce sneak paths, e.g., grounding inactive rows. In this paper, it is assumed that these approaches are used.

B. IMPLY in a Crossbar

The IMPLY logic gate can be integrated inside a crossbar, where P and Q are two memristors in the same row within the crossbar. The voltages V_{SET} and V_{COND} are the voltages of the word line, and the bit line is connected to a resistor R_G. To compute different Boolean functions with more than two memristors, the memristors are placed within the same row within the crossbar. Since the IMPLY operation is destructive to P and Q, if the data of the input to P is significant, a copy is assigned to a designated memristor. A schematic of a crossbar-based IMPLY logic gate is shown in Fig. 11.

V. Logic Gate Design Methodology

In this section, design considerations and constraints for a memristor-based IMPLY logic gate in a crossbar are described.

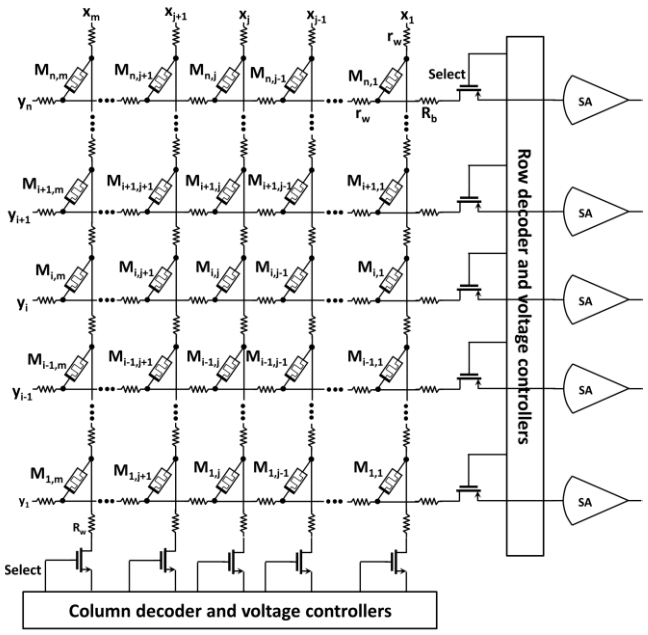


Fig. 10. $m \times n$ memristive crossbar. The columns show the word lines and the rows identify the bit lines. Each $M-ij$ is a memristor. The resistance of the conductive line is nr_w for the column line and mr_w for the row line. R_w and R_b are, respectively, the word and bit line resistance.

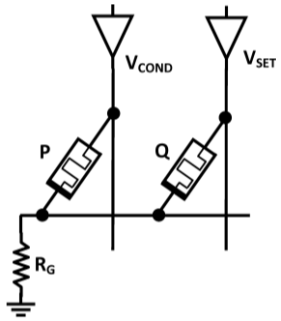


Fig. 11. IMPLY logic gate inside a memristor-based crossbar.

It is assumed that the memristor behavior is deterministic, rather than stochastic.

A. Design Flow and Constraints

Although no complete and accurate memristor model yet exists, all of the proposed memristor models are relatively complicated and the exact behavior of a memristive logic circuit is therefore mathematically cumbersome. A need therefore exists for heuristics for designing memristive circuits. For memristor-based IMPLY logic gates, the appropriate circuit parameters (R_G,V_{SET}, V_{COND}, and the time to apply the voltages T) need to be determined under some general constraints. These constraints include minimizing power consumption (only dynamic power consumption in a memristor-based crossbar), reducing area (the number of active memristors in a crossbar and the number of transistors in the controller), lowering the delay time of the logic gate, and increasing the robustness of the circuit (by reducing resistance drift during operation for those input cases where the logical output does not change). The parasitic capacitance of the CMOS transistors connected to the crossbar and the parasitic resistance of the metal lines as well as the sneak path phenomenon also need to be considered.

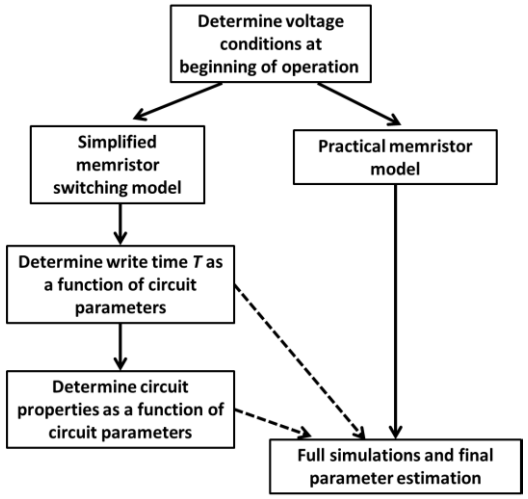


Fig. 12. Design flow for memristor-based IMPLY logic gates.

TABLE II
INPUT GATE VOLTAGES V_Q AND V_P, RESPECTIVELY, AT MEMRISTORS P AND Q AT $t = 0$, UNDER THE ASSUMPTIONS THAT THE MEMRISTANCE OF LOGIC ONE AND LOGIC ZERO IS, RESPECTIVELY, R_{ON} AND R_{OFF}, WHERE $R_{\mathrm{OFF}} \gg R_{\mathrm{ON}}$

Case	$V_Q(t=0)$	$V_P(t=0)$
1	$\frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_G}{R_{OFF}+2R_G}\cdot V_{COND}$	$-\left[\frac{R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{COND}\right]$
2	$V_{SET}\cdot\frac{R_{ON}}{R_{OFF}}\cdot\frac{R_{OFF}+R_G}{R_{ON}+R_G} \approx V_{SET}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+R_G} - V_{COND}\right]$
3	$V_{SET} - V_{COND}\cdot\frac{R_G}{R_{ON}+R_G}$	V_{COND}
4	$V_{SET}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_G}{R_{ON}+2R_G}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G}\right]$

A general flow for the design of a memristor-based IMPLY logic gate is shown in Fig. 12. The design of a general Boolean function is demonstrated through a case study in Section VI. After determining the topology of the circuit, the conditions at the beginning of operation need to be determined. These static conditions do not depend on the memristor model and provide necessary conditions for correct circuit behavior. Simplified memristor models use several heuristics to approximate the circuit characteristics. The TEAM model [23] is used here to estimate the circuit parameters.

B. Design Constraints and Parameter Determination for IMPLY Logic Gate

In the design of a basic IMPLY logic gate, the circuit parameters V_{SET}, V_{COND}, and R_G and the time to apply the voltages T need to be determined. The memristor parameters (R_{ON}, R_{OFF}, k_{ON}, k_{OFF}, α_{ON}, α_{OFF}, i_{ON}, and i_{OFF} in the TEAM model) are fixed for a given technology.

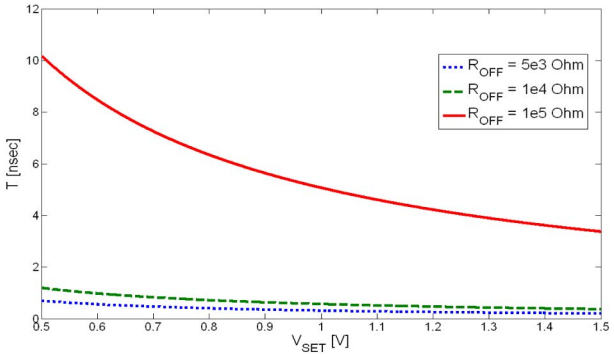


Fig. 13. Allowed write time T in case 1 for three values of R_{OFF} (5, 10, and 100 kΩ) under the assumptions of a binary resistance model and $Q' = 5 \times 10^{-14}$C.

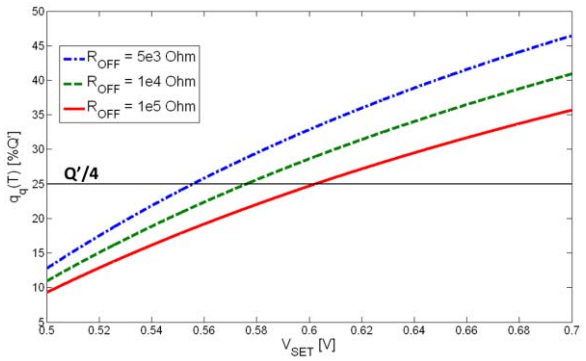


Fig. 14. Allowed values of V_{SET} for limited state drift in case 3 of $Q'/4$. V_{SET} is allowed if $q_q(T)$ is smaller than $Q'/4$ (horizontal line).

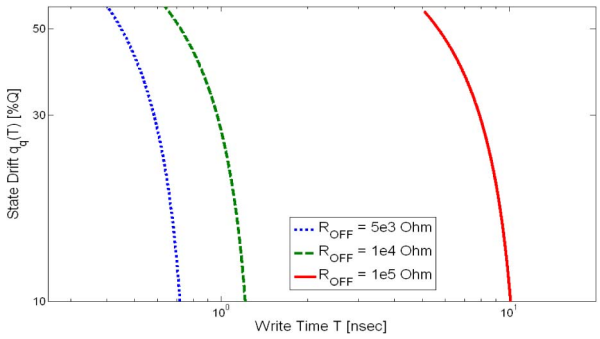


Fig. 15. Tradeoff between the speed (write time) and robustness (the state drift in case 3 for memristor Q) for three values of R_{OFF} (5, 10, and 100 kΩ) under the assumptions of a binary resistance model and $Q' = 5 \times 10^{-14}$C.

Although difficult to compute the time evolution of the voltage at Q (Fig. 4), it is possible to determine the voltage at Q at the beginning of the logic gate activity. The initial applied voltage at Q is different for each input case (a different initial memristance for Q and P). The initial voltages at P and Q are listed in Table II under the assumptions that the memristance of the logic one and logic zero is, respectively, R_{ON} and R_{OFF}, where $R_{\text{OFF}} \gg R_{\text{ON}}$.

From the initial applied voltages, some necessary conditions for correct logic behavior can be determined. The basic design principle is that the write (delay) time of the logic gate is determined from input case 1 (see Table II), but the circuit should also not exceed a specific state drift in input case 3.

A useful switching model is a binary memristance model [28]. Assume only two allowed memristances, R_{ON} and R_{OFF}. A total charge Q' flows through the memristor to cause the memristance R_{OFF} to switch to memristance R_{ON}. Under these assumptions and by solving both the switching behavior in case 1 and the write time T as a function of Q', the circuit parameter T is

$$T = \left[\frac{R_{\text{OFF}}^2 + 2R_{\text{OFF}}R_G}{R_{\text{OFF}}V_{\text{SET}} + R_G\left[V_{\text{SET}} - V_{\text{COND}}\right]}\right] \cdot Q'. \tag{5}$$

The write time for different circuit parameters and varying V_{SET} is shown in Fig. 13. Note that the logic gate is faster with a higher applied voltage or a smaller R_{OFF}.

Under this model, it is possible to limit the state drift (case 3 in Table II) for a fixed drift. The state drift is

$$q_q(T) \approx \left[V_{\text{SET}} - \frac{R_G}{R_{\text{ON}} + R_G}V_{\text{COND}}\right] \cdot \left[\frac{R_{\text{OFF}} + 2R_G}{R_{\text{OFF}}V_{\text{SET}} + R_G\left[V_{\text{SET}} - V_{\text{COND}}\right]}\right] \cdot Q' \tag{6}$$

where $q_q(T)$ is the total charge flowing through memristor Q after time T, as in case 3. If the state drift is limited to a value of $Q'/4$ as the maximum state drift, after four executions of the logic gate in case 3 the state drift would change the memristive logic state of q. This phenomenon requires a refresh every three executions of the logic gate since the logic state would change to an invert value during the fourth time. The allowed value of V_{SET} for several circuit parameters is shown in Fig. 14. Note that the state drift is more significant with a higher applied voltage, or with a smaller R_{OFF}. Combining Figs. 13 and 14, the tradeoff between the speed and robustness of a memristive IMPLY logic gate is illustrated in Fig. 15.

Another simple and useful memristor model assumes nonlinear behavior with a fixed threshold voltage V_{ON} [28]. Under this model, for an applied voltage below V_{ON}, the memristance is unchanged. To produce correct logical behavior, the initial applied voltage on Q must be above the threshold voltage in case 1 and below the threshold voltage in case 3. Adding this assumption to the initial applied voltage (see Table II) leads to the following two conditions on the circuit parameters:

$$R_{\text{ON}} \cdot \frac{V_{\text{SET}} - V_{\text{ON}}}{V_{\text{ON}} - \left[V_{\text{SET}} - V_{\text{COND}}\right]} < R_G < R_{\text{OFF}} \cdot \frac{V_{\text{SET}} - V_{\text{ON}}}{2V_{\text{ON}} - \left[V_{\text{SET}} - V_{\text{COND}}\right]} \tag{7}$$

$$\frac{V_{\text{SET}}}{V_{\text{COND}}} < \frac{R_{\text{OFF}}}{R_{\text{ON}}}. \tag{8}$$

The allowed value for R_G for several circuit parameters with varying V_{SET} is shown in Fig. 16. A reasonable value of R_G is the geometric mean of R_{ON} and R_{OFF}

$$R_G = \sqrt{R_{\text{ON}} \cdot R_{\text{OFF}}} \tag{9}$$

to maintain a constant ratio between each pair of resistances, R_{ON} and R_G, and R_G and R_{OFF}. Other values of R_G are also possible.

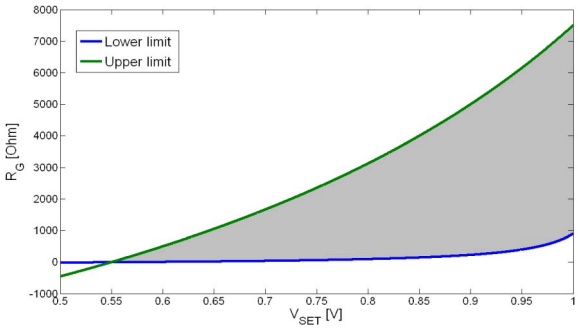


Fig. 16. Allowed value of R_G depends on V_{SET}. The upper line is the upper bound for allowed R_G and the lower line is the lower allowed bound for R_G. Under the assumption of a threshold voltage $V_{\text{ON}} = 0.55$ V, $V_{\text{COND}} = 0.5$ V, $R_{\text{ON}} = 100\ \Omega$, and $R_{\text{OFF}} = 10$ kΩ.

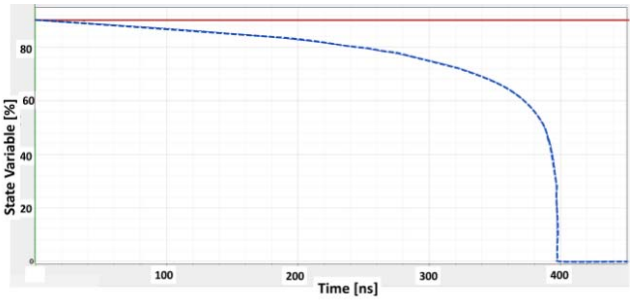


Fig. 17. State variable of q when applying an IMPLY logic gate for cases 1 (dashed line) and 3 (solid line). The parameters of the circuit are $V_{\text{SET}} = 1$ V, $V_{\text{COND}} = 0.5$ V, and $R_G = 10$ kΩ. The parameters of the memristors are $k_{\text{ON}} = 0.05$, $i_{\text{ON}} = 7\mu$A, and $\alpha_{\text{ON}} = 3$. The delay of the IMPLY logic gate is 397.1 ns and the state drift is 0.0007%, equivalent to 145,000 executions before the need to refresh.

C. Example of 1-Bit IMPLY Logic Gate

As a specific example of applying the flow chart of Fig. 12, assume the requirement is a maximum write time (delay) of 0.5 μsec. Note that the actual write time of a practical memristor is significantly faster [25]. The maximum allowed state drift is 0.00001 R_{OFF} (0.001% of the state drift as compared to full switching, equivalent to 10^5 executions of the logic gate before completely switching).

Assume a memristor with R_{ON} and R_{OFF}, respectively, of 1 and 100 kΩ. Set one circuit parameter V_{COND} to 0.5 V. From Figs. 13 and 14, note that as V_{SET} rises, the logic gate write time T decreases and the gate response is faster; however, the state drift phenomenon is more significant. From (8)

$$0.5\ \text{V} < V_{\text{SET}} < 50\ \text{V}. \tag{10}$$

This expression only produces a lower bound on V_{SET}, since the upper bound is significantly higher than practical on-chip supply voltages. For a current-controlled memristor (e.g., TEAM model), it is unrealistic to determine an exact equivalent voltage threshold (which depends on the transient memristance of the device). A sufficient approximation for an equivalent threshold voltage is

$$V_{\text{ON}} = i_{\text{ON}} \cdot R_{\text{OFF}} \tag{11}$$

where V_{ON} is the voltage threshold, and i_{ON} is the current threshold. For a memristor with a current threshold of 7 μA, the equivalent voltage threshold is 0.7 volts. From (7), R_G is

$$1.5\ \text{k}\Omega < R_G < 33.3\ \text{k}\Omega. \tag{12}$$

The widely used linear ion drift memristor model [12], [23] is incompatible with IMPLY logic gates. In this model, the memristance changes linearly for any applied voltage; the state drift phenomenon is therefore significant and intolerable for IMPLY logic gates [28]. Hence, a different memristor model with a current threshold, such as the TEAM model [23], is preferable. The TEAM model accurately describes the physical behavior of memristors. The chosen circuit parameters for this example are $R_{\text{ON}} = 1$ kΩ, $R_{\text{OFF}} = 100$ kΩ, $V_{\text{COND}} = 0.5$ V, $V_{\text{SET}} = 1$ V, and $R_G = 10$ kΩ. SPICE simulation based on these parameters for the memristance of q are shown in Fig. 17, where the write time (delay) of this logic gate is 397.1 ns and the state drift is 0.00069%, equivalent to about 145,000 executions before switching.

The write time (delay) and state drift for varying R_G and V_{SET} are listed in Tables III and IV. An increase in the resistance of R_G or decrease in the voltage level of V_{SET} increases the delay of the gate, but lowers the state drift phenomenon (and vice versa). The write time (delay) and state drift for different memristor parameters are listed in Table IV. An increase in the nonlinearity of the memristors (α_{ON}) increases the delay of the gate, but lowers the state drift phenomenon (and vice versa). An increase in k_{ON} decreases the delay of the gate without changing the state drift phenomenon.

TABLE III
WRITE TIME AND STATE DRIFT FOR DIFFERENT VALUES OF R_G. ALL VALUES SATISFY (10) AND (12). V_{COND} IS SET TO 0.5 V, $K_{\text{ON}} = 0.05$, $I_{\text{ON}} = 7\ \mu$A, AND $\alpha_{\text{ON}} = 3$

R_G [kΩ]	T [μsec]	State Drift [%R_{OFF}]	Writes Before Refresh [#]
1	0.1307	0.4655	215
3.5	0.1782	0.00244	4.09E4
5	0.2144	0.00184	5.43E4
10	0.3971	0.00069	1.45E5
15	0.7472	0.0009	1.15E6
17.5	1.038	0.00001	1.743E7
20	1.46	0	∞
30	3.063	0	∞

TABLE IV
WRITE TIME AND STATE DRIFT FOR DIFFERENT VALUES OF V_{SET} AND MEMRISTOR PARAMETERS. ALL VALUES SATISFY (19) AND (12). USING THE SAME DEFAULT VALUES AS TABLE III. $R_G = 10$ KΩ

Parameter	T [μsec]	State Drift [%R_{OFF}]	Writes Before Refresh [#]
Base	0.3971	0.00069	1.45E5
$V_{SET} = 1.2$ V	0.0945	0.31208	320
$k_{on} = 0.1$	0.1986	0.00069	1.45E5
$k_{on} = 0.01$	1.9866	0.0007	1.44E5
$\alpha_{on} = 1$	0.1587	0.3669	273
$\alpha_{on} = 4$	0.7927	0.0004	2.52E5

TABLE V
RESISTANCE OF A CMOS DRIVER FOR 0.12 μm CMOS PROCESS

W [µm]	W/L	CMOS Driver Resistance [Ω]	Voltage Drop with a Load of 100 kΩ
0.13	1	12.8k	11.33%
0.3	2.3	6.4k	6.00%
0.5	3.8	3.8k	3.67%
0.75	5.8	2.5k	2.42%
1	7.7	1.8k	1.83%
1.3	10	1.4k	1.33%
2.5	19.2	708	0.67%
5	38.5	349	0.33%
10	76.9	173	0.17%
20	153.8	86	0.08%

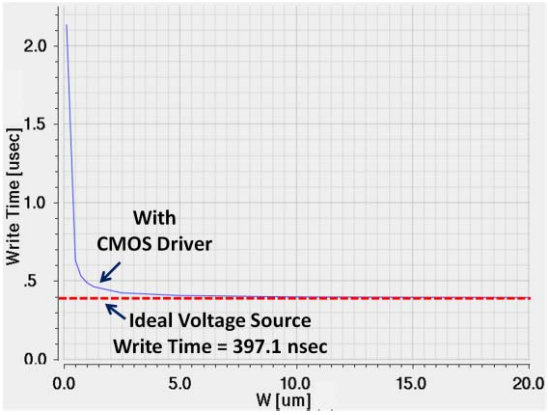


Fig. 18. Write time of an IMPLY logic gate with CMOS drivers for various CMOS widths (solid blue line) as compared to the write time with ideal voltage source (dashed red line). A 0.12 μm CMOS process is used; other circuit parameters are the same as in Fig. 17.

D. Variations in V_{SET} and V_{COND}

In previous sections, it is assumed that ideal voltage sources are used for V_{SET} and V_{COND}. Practical implementations, however, suffer from variations in the voltage level, mainly due to the resistance of the CMOS drivers. The CMOS drivers add resistance in series with the circuit and change the applied voltages. These voltage drops change the performance (as determined from input case 1) and the state drift (as determined from input case 3).

To evaluate the influence of CMOS drivers on performance and state drift, the IMPLY logic gate is simulated with similar circuit parameters as in Section V-C. The equivalent resistance of the CMOS driver for various CMOS widths is listed in Table V. The write time for different driver widths is shown in Fig. 18. For a W/L ratio of 10, the write time of the IMPLY logic gate with CMOS drivers increases by approximately 15%, as compared to ideal voltage sources. For a W/L ratio of 75, the increase in the write time is negligible (less than 1%).

To evaluate the change in the state drift phenomenon, the IMPLY logic gate is evaluated for input case 3. The difference in the state drift is listed in Table VI, exhibiting negligible difference for all W/L ratios. To overcome variations in the voltage source, the applied voltages (V_{SET} and V_{COND}) can be increased. Alternatively, the resistance of the circuit can be increased, by increasing R_G or using memristors with higher R_{ON} and R_{OFF} (e.g., the memristors in [42] have R_{ON} of approximately 300 kΩ), or the resistance of the CMOS driver can be decreased by increasing the W/L ratio.

TABLE VI
STATE DRIFT OF THE IMPLY LOGIC GATE WITH CMOS BUFFERS AS COMPARED TO IDEAL VOLTAGE SOURCES FOR VARIOUS W/L RATIO

W [µm]	W/L	Difference in the State
0.13	1	-0.000502%
0.3	2.3	-0.000150%
0.5	3.8	0.000009%
0.75	5.8	0.000053%
1	7.7	0.000059%
1.3	10	0.000056%
2.5	19.2	0.000038%
5	38.5	0.000021%
10	76.9	0.000011%
20	153.8	0.000006%

VI. 8-BIT IMPLY FULL ADDER: A CASE STUDY

IMPLY together with FALSE (the function that always yields zero as an output) provide a complete logical structure. While any Boolean function can be executed, an efficient procedure is required to reduce the area and computational time. In this section, a case study of an 8-bit full adder is presented to discuss several design constraints and issues for general Boolean functions. In this case study, three approaches are considered: a general algorithm [29] is considered first, which requires a long sequence and only two additional memristors. Two other specific approaches–serial and parallel–are also considered. These approaches significantly reduce the required sequence of operational steps, where the parallel approach requires more memristors for faster execution as compared to the serial approach.

A. General Boolean Functions

An algorithm to implement any general Boolean function using only IMPLY and FALSE has been proposed in [29]. This algorithm requires $n+3$ memristors for any general Boolean function $f:B^n \rightarrow B$. While this algorithm is efficient in terms of area (the number of memristors to compute a function), it is inefficient in terms of computational time and requires $O(2^{kn})$ computational steps, where n is the number of input memristors and k is the number of additional functional memristors for the computational process. A different approach is therefore required to improve the computational time. This new approach is demonstrated in this section through a case study.

Several Boolean functions being implemented by IMPLY and FALSE are listed in Table VII. These functions are the basic building blocks of any general Boolean function. Choosing the proper building blocks and computing sequence are key when the objective is to minimize the number of

TABLE VII
BASIC BOOLEAN OPERATIONS BASED ONLY ON IMPLY AND FALSE

Structure	*Operation*	*Comments*
$0 \rightarrow q$	q' = 1	
$1 \rightarrow q$	q' = q	
$p \rightarrow 0$	q' = NOT(p)	
$(A \rightarrow (B \rightarrow 0)) \rightarrow 0$	q' = A AND B	Result in different memristor than the inputs
$(A \rightarrow 0) \rightarrow B$	B' = A OR B	
$(A \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow 0)$	q' = A XOR B	Requires copying of the inputs, separate output q
FALSE(B), FALSE(C), $A \rightarrow C$, $C \rightarrow B$	B' = A	Copy operation – copy A to B

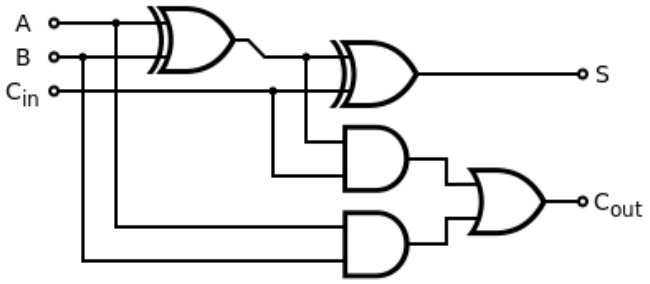


Fig. 19. Full adder consisting of two XOR gates, two AND gates, and an OR gate.

computational steps and memristors. To reduce the number of computational steps, parallelism can be exploited, where several IMPLY and FALSE operations occur during the same clock cycle. Since the operation is accomplished within the crossbar structure, the topology of the entire array needs to be considered, including possible sneak paths. Other methods for parallelism that do not suffer from sneak paths use unipolar memristors or, alternatively, insert switches between rows, which deviates from the crossbar structure. Modifying the crossbar structure to parallelize the execution is discussed in Section VI.

It is sometimes necessary to copy the value from a memory cell to other cells. The copy operation is also required when data is used multiple times, since the destruction of the input is undesired, or there is a need to transfer data to different rows within the crossbar. The copy operation is also listed in Table VII.

B. CMOS Full Adder

The input of the full adder are two 8-bit numbers and the output is one 8-bit number $S_7, S_6, \ldots, S_0$ and 1-bit carry C_{out}. The basic structure of a CMOS 8-bit ripple carry adder consists of eight full adders, where the logical operation of each adder is

$$S_i = A_i \oplus B_i \oplus C_i \tag{13}$$

$$C_{\text{out}} = (A_i \cdot B_i) + (C_i \cdot (A_i \oplus B_i)). \tag{14}$$

A single CMOS 8-bit adder consists of 400 CMOS transistors, as shown in Fig. 19 for a basic full adder.

TABLE VIII
COMPARISON OF N-BIT FULL ADDERS. THE NUMBERS IN THE BRACKETS ARE FOR AN 8-BIT FULL ADDER

		Base [29]	Optimized Approaches	
			Serial	Parallel
Execution steps		89*N* (712)	29*N* (232)	5*N*+18 (58)
Memristors	Input	2*N*	2*N*	2*N*
	Output	*N*+1	*N*+1	*N*+1
	Functional	4	2	6*N*-1
	Total	3*N*+5 (29)	3*N*+3 (27)	9*N* (72)
Special functions required	Parallel FALSE	-	-	V
	IMPLY between lines	-	-	V
	TRUE	V	-	-

C. IMPLY Full Adder

Several approaches exist to design an 8-bit full adder based solely on IMPLY and FALSE operations. The basic approach is to follow the algorithm proposed in [29]. Two additional approaches are considered–serial and parallel. To evaluate these approaches, the total number of memristors and the number of computation steps are compared. The general algorithm from [29] requires 712 computational steps, while the serial approach lowers the computational time to 232 computational steps with approximately the same number of memristors, and the parallel approach has the best performance of 58 computational steps but requires double the number of memristors. A comparison among the approaches is listed in Table VIII.

To execute a XOR operation, two functional memristors *M1* and *M2* are required, where the complete sequence, as listed in Table VII, is

$$\begin{aligned} A \text{ XOR } B: \ & \text{FALSE}(M1), \text{FALSE}(S), A \rightarrow S, S \rightarrow M1 \\ & \text{FALSE}(M2), \text{FALSE}(S), B \rightarrow S, S \rightarrow M2 \\ & B \rightarrow M1, \text{FALSE}(S), M1 \rightarrow S \\ & A \rightarrow M2, M2 \rightarrow S. \end{aligned}$$

The first two rows are copy operations of A and B, respectively, to *M1* and *M2* since the IMPLY operation destroys both inputs. To execute S_i, the execution process is divided into two XOR operations, where (13) is

$$S_i = (A_i \oplus B_i) \oplus C_i. \tag{15}$$

This execution requires two functional memristors and 26 computational steps for S_i, while the intermediate XOR of A_i and B_i is also used for $C_{\text{out},i}$, where (14) becomes

$$\begin{aligned} C_{\text{out},i} = & \left(A_i \rightarrow \left(B_i \rightarrow' 0'\right)\right) \\ & \rightarrow \left(\left(C_i \rightarrow \left(\left(A_i \oplus B_i\right) \rightarrow' 0'\right)\right) \rightarrow' 0'\right). \end{aligned} \tag{16}$$

Several possible sequences exist for executing C_i using three functional memristors to decrease the number of computational steps. Furthermore, A_i, B_i, and C_i can also be treated as functional memristors after the initial value is changed during the execution process. The complete sequence is described in the supplementary material.

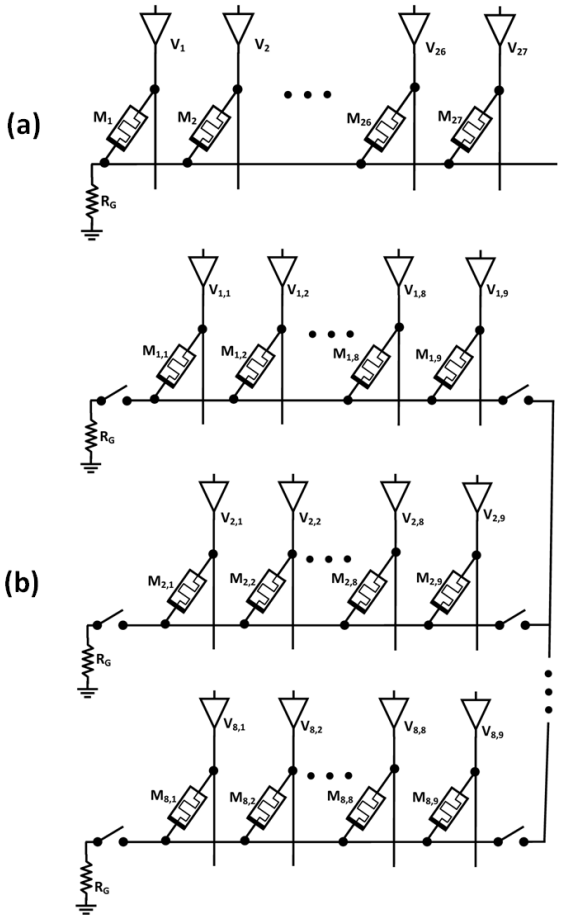


Fig. 20. 8-bit full adder for (a) serial approach and (b) parallel approach. For the serial approach, 27 memristors are used in the same row of a standard crossbar structure. The parallel approach requires a more complex crossbar structure, where a switched connection between rows exists. Each bit execution is done in a different row using nine memristors.

For an 8-bit full adder, two approaches have been examined in the case study. The serial approach executes one operation every clock cycle–IMPLY or FALSE. For the serial approach, all memristors are in the same row, as shown in Fig. 20(a). In the parallel approach, independent operations are executed during the same clock cycle, reducing the number of required computational stages. For the parallel approach, each bit in the full adder is in a different row, as shown in Fig. 20(b). The carry is passed between the different rows and the FALSE operations are simultaneously completed for several memristors. The parallel approach requires some modifications which differ from the crossbar structure, adding connections between the rows of the crossbar. These modifications also eliminate the sneak path phenomenon while increasing the area as compared to a conventional crossbar.

VII. Beyond von Neumann: Logic Inside the Memory

IMPLY logic is a natural method to execute logical operations within the memristors. Memristor-based IMPLY logic has the same crossbar structure as a memristor-based memory and therefore enables the capability of performing logic operations inside the memory with the same cells used to store data. This combination enables innovative computing architectures, rather than the classical von Neumann architecture where the computing operations and the data storage are separated.

For these novel architectures, part of the computation is achieved inside the memory, with no separation with the data read and write operations. These architectures are particularly appropriate for massive parallel applications, where vast amount of data need to be processed. In von Neumann architecture for massive parallel applications, the data transfer requires a wide data bus, long latency, and consumes relatively high power. In these novel architectures, the memory and logical operations are in the same crossbar structure, almost no data transfer is required, and the latency and power are significantly reduced, although the memristor IMPLY logic delay is greater than the CMOS logic delay.

In these innovative architectures, the memristive memory serves two roles–as memory to store data and as a computational unit. The function of a specific memristor can be decided dynamically. Each memristor can act as either a memory cell or as part of an IMPLY logic gate in different stages of the operation. The effective size of the memory and the computational unit is flexible and can vary for different applications. A memristor-based memory requires a relatively complex controller that behaves as a regular memory controller and also sends control signals (V_{SET} and V_{COND}) to the IMPLY logic gates. This novel architecture requires a new instruction set, requiring specific instructions for logic operations inside the memory.

VIII. Conclusion

An IMPLY logic gate is a natural way to perform logic operations with memristors. This logic gate can be integrated within a memristor-based memory and, together with FALSE, provide a complete logic family. This memristive logic gate also enables non-von Neumann architectures, which may open a new era in computer architecture.

The potential benefits of memristive circuits in terms of density and power support further work in this field. The results described in this paper can be used to direct further research on device structure optimization, logic synthesis methods, array structures, and computing architectures.

References

[1] L. O. Chua, "Memristor—The missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.

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

[3] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, pp. 888–900, Jun. 2005.

[4] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Hybrid CMOS-memristor logic," *IEEE Trans. Very Large Scale Integr. (VLSI)*, in preparation.

[5] M. Klimo and O. Such, *Memristors Can Implement Fuzzy Logic*. Ithaca, NY, USA: Cornell Univ. Press, Oct. 2011.

[6] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Mater. Sci. Process.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.

[7] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, pp. 873–876, Apr. 2010.

[8] Y. V. Pershin and M. Di Ventra, "Neuromorphic, digital and quantum computation with memory circuit elements," *Proc. IEEE*, vol. 100, no. 6, pp. 2071–2080, Jun. 2012.

[9] S. Shin, K. Kim, and S.-M. Kang, "Reconfigurable stateful NOR gate for large-scale logic-array integrations," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 7, pp. 442–446, Jul. 2011.

[10] D. Biolek, Z. Biolek, and V. Biolkova, "Pinched hysteresis loops of ideal memristors, memcapacitors, and meminductors must be 'self-crossing'," *Electron. Lett.*, vol. 47, no. 25, pp. 1385–1387, Dec. 2011.

[11] L. O. Chua, "Resistance switching memories are memristors," *Appl. Phys. A, Mater. Sci. Process.*, vol. 102, no. 4, pp. 765–783, Mar. 2011.

[12] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[13] D. Sacchetto, M. H. Ben-Jamaa, S. Carrara, G. DeMicheli, and Y. Leblebici, "Memristive devices fabricated with silicon nanowire Schottky barrier transistors," in *Proc. IEEE Int. Symp. Circuits Syst.*, May/Jun. 2010, pp. 9–12.

[14] K. A. Campbell, A. Oblea, and A. Timilsina, "Compact method for modeling and simulation of memristor devices: Ion Conductor Chalcogenide-based Memristor Devices," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architect.*, Jun. 2010, pp. 1–4.

[15] X. Wang, Y. Chen, H. Xi, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE Electron Device Lett.*, vol. 30, no. 3, pp. 294–297, Mar. 2009.

[16] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, Jun. 2009.

[17] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A versatile memristor model with non-linear dopant kinetics," *IEEE Trans. Electron Devices*, vol. 58, no. 9, pp. 3099–3105, Sep. 2011.

[18] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, pp. 429–433, Jul. 2008.

[19] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *Proc. 12th Int. Workshop Cellular Nanosc. Netw. Appl.*, Feb. 2010, pp. 1–4.

[20] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *J. Appl. Phys.*, vol. 106, no. 7, pp. 1–6, Oct. 2009.

[21] H. Abdalla and M. D. Pickett, "SPICE modeling of memristors," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 1832–1835.

[22] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE Electron Device Lett.*, vol. 32, no. 10, pp. 1436–1438, Oct. 2011.

[23] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM—ThrEshold adaptive memristor model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 1, pp. 211–221, Jan. 2013.

[24] J. Borghetti, Z. Li, J. Strasnicky, X. Li, D. A. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proc. Nat. Acad. Sci. United States Amer.*, vol. 106, no. 6, pp. 1699–1703, Feb. 2009.

[25] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, no. 48, pp. 1–7, Nov. 2011.

[26] J. J. Yang, M.-X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams, "High switching endurance in TaO_x memristive devices," *Appl. Phys. Lett.*, vol. 97, no. 23, pp. 1–3, Dec. 2010.

[27] J. Nickel, "Memristor materials engineering: From flash replacement towards a universal memory," in *Proc. IEEE IEDM Adv. Memory Technol. Workshop*, Dec. 2011, pp. 1–3.

[28] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY logic design procedure," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2011, pp. 142–147.

[29] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2009, pp. 33–36.

[30] E. Lehtonen, J. H. Poikonen, and M. Laiho, "Two memristors suffice to compute all boolean functions," *Electron. Lett.*, vol. 46, no. 3, pp. 239–240, Feb. 2010.

[31] S. Shin, K. Kim, and S.-M. Kang, "Reconfigurable stateful NOR gate for large-scale logic-array integrations," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 7, pp. 442–446, Jul. 2011.

[32] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Mater.*, vol. 9, no. 5, pp. 403–406, Apr. 2010.

[33] A. Flocke and T. G. Noll, "Fundamental analysis of resistive nano-crossbars for the use in hybrid Nano/CMOS-memory," in *Proc. Eur. Solid State Circuits Conf.*, Sep. 2007, pp. 328–331.

[34] M. A. Zidan and K. N. Salama, "Memristor based memory: The sneak paths problem and solutions," *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, Feb. 2013.

[35] C. A. David and B. Feldman, "High-speed fixed memories using large-scale integrated resistor matrices," *IEEE Trans. Comput.*, vol. 17, no. 8, pp. 721–728, Aug. 1968.

[36] W. T. Lynch, "Worst-case analysis of a resistor memory matrix," *IEEE Trans. Comput.*, vol. 18, no. 10, pp. 940–942, Oct. 1969.

[37] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proc. IEEE*, vol. 100, no. 6, pp. 2021–2032, Jun. 2012.

[38] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 1–5.

[39] O. Kavehei, S. Al-Sarawi, K.-R. Cho, K. Eshraghian, and D. Abbot, "An analytical approach for memristive nanoarchitectures," *IEEE Trans. Nanotechnol.*, vol. 11, no. 2, pp. 374–385, Mar. 2012.

[40] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, P. Crozat, N. Zerounian, J.-V. Kim, C. Chappert, and H. Ohno, "Single-shot time-resolved measurement of nanosecond-scale spin-transfer induced switching: Stochastic versus deterministic aspects," *Phys. Rev. Lett.*, vol. 100, no. 5, pp. 057206-1–057206-4, Feb. 2008.

[41] R. Soni, P. Meuffels, G. Staikov, R. Weng, C. Kügeler, A. Petraru, M. Hambe, R. Waser, and H. Kohlstedt, "On the stochastic nature of resistive switching in Cu doped $Ge_{0.3}Se_{0.7}$ based memory devices," *J. Appl. Phys.*, vol. 110, no. 5, pp. 054509-1–054509-10, Sep. 2011.

[42] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba, and W. Lu, "Synaptic behaviors and modeling of metal oxide memristive device," *Appl. Phys. A*, vol. 102, no. 4, pp. 857–863, Feb. 2011.

Shahar Kvatinsky (S'12) received the B.Sc. degree in computer engineering and applied physics and the M.B.A. degree from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2009 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, Technion-Israel Institute of Technology, Haifa, Israel.

Prior to his Ph.D. studies, he worked for Intel as a circuit designer.

Guy Satat received the B.Sc. degree in electrical engineering and the B.Sc. degree in physics from the Technion-Israel Institute of Technology, Haifa, Israel, as part of the Technion's Program for excellent students.

He joined Intel, Inc., in 2011, and worked on interconnect architecture. In 2013, he joined the Media Laboratory, Camera Culture Group, Massachusetts Institute of Technology, Cambridge, MA, USA, as a Graduate Student, and worked on ultrafast imaging and health imaging.

Nimrod Wald received the B.Sc. degree in electrical engineering and physics from Technion-Israel Institute of Technology, Haifa, Israel, in 2013.

He joined Qualcomm, Inc., San Diego, CA, USA, in 2011, as a Hardware Designer, and he has been a Hardware Architect since 2013 in the area of performance analysis.

Eby G. Friedman (M'79–SM'90–F'00) received the B.S. degree from Lafayette College, Easton, PA, USA, in 1979, and the M.S. and Ph.D. degrees from the University of California, Irvine, CA, USA, in 1981 and 1989, respectively, all in electrical engineering.

He was with Hughes Aircraft Company, Glendale, CA, USA, from 1979 to 1991, rising to the position of manager of the Signal Processing Design and Test Department, responsible for the design and test of high performance digital and analog IC's. He has been with the Department of Electrical and Computer Engineering at the University of Rochester, Rochester, NY, USA, since 1991, where he is a Distinguished Professor, and the Director of the High Performance VLSI/IC Design and Analysis Laboratory. He is also a Visiting Professor with the Technion-Israel Institute of Technology. His current research interests include high performance synchronous digital and mixed-signal microelectronic design and analysis with application to high speed portable processors and low power wireless communications.

Dr. Friedman is the author of over 400 papers and book chapters, 12 patents, and the author or editor of 16 books in the fields of high speed and low power CMOS design techniques, 3-D design methodologies, high speed interconnect, and the theory and application of synchronous clock and power delivery. He is the Regional Editor of the *Journal of Circuits, Systems and Computers*, a member of the editorial boards of the *Analog Integrated Circuits and Signal Processing*, *Microelectronics Journal*, *Journal of Low Power Electronics*, *Journal of Low Power Electronics and Applications*, and IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, Chair of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS steering committee, and a member of the technical program committee of a number of conferences. He previously was the Editor-in-Chief of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, a member of the editorial board of the Proceedings of the IEEE, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, and Journal of Signal Processing Systems, a Member of the Circuits and Systems (CAS) Society Board of Governors, Program and Technical chair of several IEEE conferences, and a recipient of the University of IEEE CAS Charles A. Dosoer Technical Achievement Award, Rochester Graduate Teaching Award, and a College of Engineering Teaching Excellence Award. He is a Senior Fulbright Fellow.

Avinoam Kolodny (SM'11) received the Doctoral degree in microelectronics from the Technion-Israel Institute of Technology, Haifa, Israel, in 1980.

He joined Intel Corporation, where he was engaged in Research and Development in the areas of device physics, VLSI circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Eletrical Engineering, Technion since 2000. His current research interests include interconnects in VLSI systems at both physical and architectural levels.

Uri C. Weiser (F'02) received the bachelor's and master's degrees in electrical engineering from Technion, Haifa, Israel and the Ph.D. degree in computer science from the University of Utah, Salt Lake City, UT, USA.

He is a Visiting Professor with the Electrical Engineering Department, Technion IIT, and an Advisor at numerous startups. He was with Intel from 1988 to 2006. At Intel, he initiated the definition of the first Pentium processor, drove the definition of Intel's MMX technology, invented (with A. Peleg) the Trace Cache, he co-managed and established the Intel Microprocessor Design Center, Austin, TX, USA, and later initiated an advanced media applications research activity. He was appointed Intel Fellow in 1996. He was with the Israeli Department of Defense as a Research and System Engineer and with National Semiconductor Design Center, Israel, where he led the design of the NS32532 microprocessor.

Dr. Weiser was an Associate Editor of the *IEEE Micro Magazine* from 1992 to 2004 and *Computer Architecture Letters*. He was a fellow of ACM.

MAGIC – Memristor Aided LoGIC

Shahar Kvatinsky, *Student Member*, *IEEE*, Dmitry Belousov, Slavik Liman, Guy Satat, *Student Member*, Nimrod Wald, Eby G. Friedman, *Fellow*, *IEEE*, Avinoam Kolodny, *Senior Member*, *IEEE*, and Uri C. Weiser, *Fellow*, *IEEE*

***Abstract*— Memristors are passive components with a varying resistance which depends on the previous voltage applied across the device. While memristors are naturally used as memory, memristors can also be used for other applications, including logic circuits. In this paper, a memristor only logic family, MAGIC (Memristor Aided LoGIC), is presented. In each MAGIC logic gate, memristors serve as an input with previously stored data and an additional memristor serves as an output. The topology of a MAGIC NOR gate is similar to the structure of a common memristor-based crossbar memory. A MAGIC NOR gate can therefore be placed within a memory, opening opportunities for novel non-von Neumann computer architectures. Other MAGIC gates also exist (*e.g.*, AND, OR, NOT, and NAND) and are described in the paper.**

***Index Terms*—Memristive systems, memristor, IMPLY, MAGIC, in-memory computing.**

I. Introduction

In recent years, the concept of a memristor, originally proposed by Leon Chua in 1971, has generated renewed interest. In [1], Chua proposed a fourth fundamental component in addition to the three already well known fundamental electronic components: the resistor, capacitor, and inductor. In [13], Chua and Kang extended the theory of memristors to memristive systems. Memristors and memristive devices are two-terminal devices, where the resistance of the device is changed by the electrical current, as shown in Figure 1. The resistance of the memristor is bounded by a minimum resistance R_{ON} and a maximum resistance R_{OFF}. In this paper, for simplicity, the terms memristor and memristive device are used interchangeably [14].

For almost forty years, the concept of a memristor was just theory, as no device exhibiting the behavior of a memristor had been produced. In 2008, Hewlett Packard Laboratories [2] announced that they had succeeded in producing a memristor. Since 2008, several possible applications of memristors have been presented. Nonvolatile emerging memory technologies, including Resistive RAM (RRAM) and Spin-Transfer Torque Magnetoresistive RAM (STT-MRAM), can be considered as memristors [14]. The primary application of memristors has been memory [3, 16, 22], where the resistance serves to store data. Another interesting and new application is memristor-based logic [4-5, 9-12, 17-18].

Material implication (IMPLY) as a memristor-based logic gate is presented in [5], [9], [11], and [17]. The memristor-based IMPLY logic gate is built within a memristive crossbar, the most commonly used memristive memory structure (particularly for RRAM). The stored data within the memristors are the input and output of the logic gate. This method, however, requires sequential voltage activation at different locations within the circuit. Furthermore, with IMPLY, the result is stored by one of the inputs and not a dedicated output memristor. The technique also requires additional circuit components (for example, a controller and an additional resistor within each row of the crossbar), dissipates high power, has high computational complexity, and requires complicated control circuitry.

In this paper, Memristor Aided LoGIC (MAGIC) - a method for memristive-only logic - is presented. This method does not require a complicated structure and enables stable evaluation of the gate function. Stable evaluation is achieved by applying a single voltage pulse at the gateway of the circuit. MAGIC NOR gates can also be fabricated within a crossbar, enabling computing within memory.

II. Operating Principle of MAGIC

MAGIC requires only memristors within the logic gates. The logical state in a MAGIC gate is represented as a resistance, where the high and low resistances are considered, respectively, as logical zero and one (for simplicity, the resistance of logical zero and logical one is considered, respectively, as R_{off} and R_{on}). The inputs and output of the logic gates are the logical states of the memristors. Unlike an IMPLY logic gate, separate memristors are required for the input and output. The inputs of the MAGIC gates are the initial logical state of the input memristors and the output is the final logical state of the memristor.

Manuscript received 24nd April, 2014. This work was partially supported by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and by Binational Science Foundation under Grant no. 2012139.

S. Kvatinsky, D. Belousov, S. Liman, A. Kolodny, and U. C. Weiser are with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel. (S. Kvatinsky corresponding author phone: 972-77-887-1923; fax: 972-4829-5757; e-mail: skva@tx.technion.ac.il).
N. Wald is with Qualcomm Inc., 1 Nahum Het St., Haifa 3190500, Israel. G. Satat is with the MIT Media Lab, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. E. G. Friedman is with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627, USA.

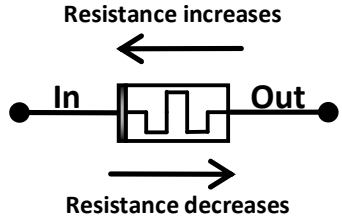


Figure 1. Memristor symbol. The polarity of the memristor is represented by a thick black line. When current flows into the device (the upper arrow), the resistance of the device increases. When current flows out of the device (the lower arrow), the resistance of the device decreases.

Operation of a MAGIC gate consists of two sequential stages. The first stage initializes the output memristor to a known logical state. In the second stage of operation, a voltage V_0 is applied across the logic gate. While applying V_0, the voltage across the output memristor depends upon the logical state of the input and output memristors. The nonlinear characteristics of the memristor, namely the threshold currents or voltages [8], are exploited to maintain correct operation. For specific input combinations, the voltage is sufficient to change the logical state of the output memristor, *i.e.*, the memristor voltage/current is greater than the threshold voltage/current, while for other input combinations, the output remains at the initialize state, *i.e.*, the memristor voltage/current is below the threshold current or voltage.

Initialization of the memristors can be achieved in several ways. For example, it is possible to use a similar topology as the circuit used in [19] for configurable memristive analog circuits, as shown in Figure 2. For MAGIC gates within memory (as described in Section III), initialization is achieved as a regular write operation within the memory cells.

In the next section, the basic MAGIC NOR gate is described. Additional MAGIC gates for different Boolean functions based on the MAGIC topology are also available, and described in Section VI.

III. MAGIC NOR Gate

A two input NOR gate consists of two input memristors (*in1*, *in2*) connected in parallel and an additional memristor (*out*) as the output. A schematic of a two input NOR gate is shown in Figure 3a. The initial execution step includes writing a low resistance into the output memristor (initialization to logical one) and, if necessary, writing the input value into memristors *in1* and *in2*. In the final execution step, the evaluation is achieved by applying a voltage pulse V_0 at the gateway of the logic gate (the gateway is defined as shown in Figure 3a).

The applied voltage produces a current that passes through the circuit and appears at memristor *out*. For the case where both input memristors are logical zero (high resistance), the voltage/current of the output memristor is lower than the memristor threshold voltage/current. Hence, the logical state of the output memristor does not change and remains at logical one. For all other input combinations, the voltage/current is greater than the memristor threshold voltage/current. The logical state of the output memristor for these input combinations switches to logical zero. The behavior of the MAGIC NOR gate is shown in Figure 3b.

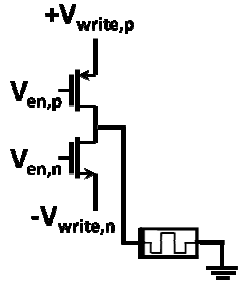


Figure 2. Example of placing a memristor into the initialization stage. This example is similar to writing a configurable analog circuit [19]. A different initialization scheme is presented in Section III for MAGIC within memory.

Assume a memristor with voltage thresholds of $V_{T,ON}$ and $V_{T,OFF}$. For correct circuit behavior, the voltage at the output memristor is lower than $V_{T,OFF}$ when both inputs are logical zero. For all other input combinations, the voltage across the output memristor should be greater than $V_{T,OFF}$. The minimum voltage at the output memristor greater than $V_{T,OFF}$ is achieved when one input is logical one and the other input is logical zero. Combining the cases where the voltage at the output memristor is above and below the threshold voltage leads to a design constraint on the applied voltage V_0. The constraint is (assuming $R_{off} >> R_{on}$)

$$2V_{T,OFF} < V_0 < \frac{R_{OFF}}{2R_{ON}} \cdot V_{T,OFF}. \quad (1)$$

When an input memristor is logical zero, the operation of a MAGIC NOR can be destructive, changing the input to logical one during execution. To eliminate destroying the input, the voltage across the input memristor needs to be below the threshold voltage $V_{T,ON}$. The maximum applied voltage for a two-input NOR gate is therefore

$$V_0 < \min\left[\frac{R_{OFF}}{2R_{ON}} \cdot V_{T,OFF}, \left|V_{T,ON}\right|\right]. \quad (2)$$

Multiple-input (three or more) NOR logic gates can also be produced in a similar manner, as shown in Figure 3c. For χ input memristors, the design constraints are

$$\frac{V_{T,OFF}}{R_{ON}} \cdot \left[R_{ON} + \left(\frac{R_{OFF}}{\chi - 1}\right) \| R_{ON}\right] < V_0 < V_{T,OFF} \cdot \left[1 + \frac{R_{OFF}}{\chi R_{ON}}\right]. \quad (3)$$

For non-destructive operation of a χ-input NOR, the maximum applied voltage is

$$V_0 < \min\left[V_{T,OFF} \cdot \left(1 + \frac{R_{OFF}}{\chi R_{ON}}\right), \left(1 + \frac{\chi R_{ON}}{R_{OFF}}\right) \cdot |V_{T,ON}|\right]. \quad (4)$$

IV. MAGIC Within a Crossbar Array

RRAM commonly utilizes a crossbar structure. The crossbar structure enables dense memory of 4 F^2, where F is the feature size. Memristive-only logic gates within a memristive crossbar reduce power and provide an opportunity for novel non-von Neumann architectures, where the logical operations are executed within the memory [11]. When performing logic within the memory, the input is the stored data within the memristors and the output is the stored data after execution. Initialization of the input and output is achieved as a regular

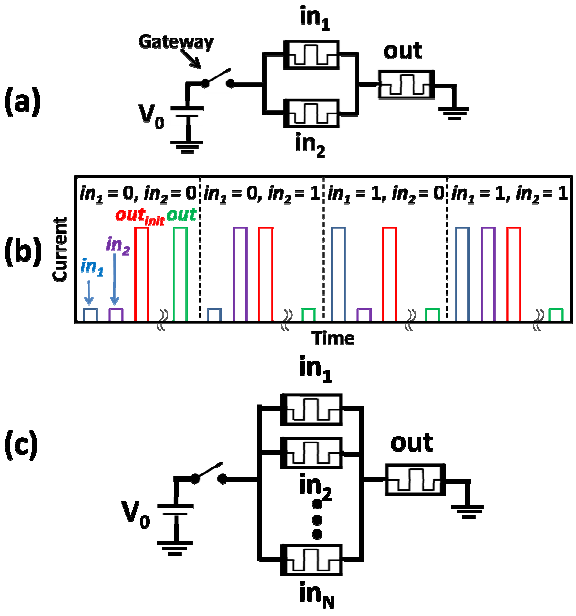


Figure 3. MAGIC NOR. (a) Schematic of a two input NOR logic gate. The logic gate consists of two input memristors *in1* and *in2*, and an output memristor *out*. During execution, a voltage V_0 is applied at the gateway of the circuit (marked by an arrow). (b) Simulations of a two input NOR gate for all input combinations. The different curves show the currents read from each memristor prior to execution and after applying V_0. (c) Schematic of an N input NOR gate.

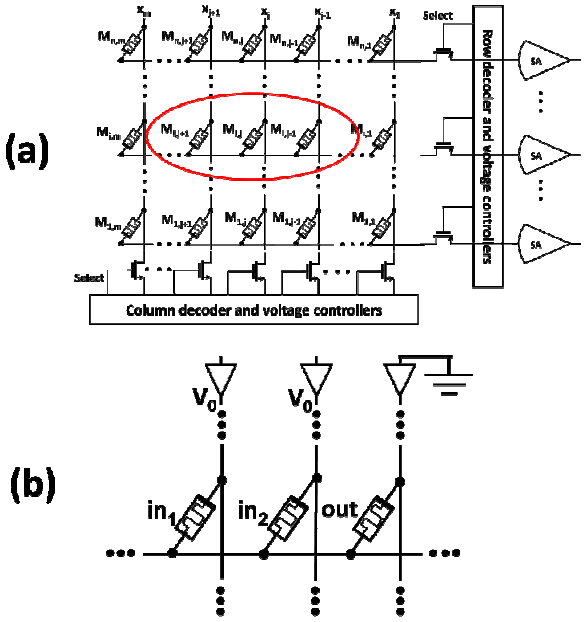


Figure 4. MAGIC NOR gate within a crossbar array. (a) Schematic of a memristive crossbar structure. A two input NOR gate is achieved in row *i*, where *in1* and *in2* are, respectively, in columns *j*+1 and *j* and *out* is in column *j*-1, as marked by an oval. (b) Schematic of a two input NOR gate within a crossbar array. The voltage at the gateway V_0 is the applied voltage at columns *j* and *j*+1, while column *j*-1 is connected to ground. Note that the schematic is identical to the figure shown in Figure 3a.

memory write operation, and sensing the result is achieved as a regular memory read operation.

To integrate a memristive-only logic gate within a crossbar array, two requirements need to be satisfied: the structure and connections of the logic gate should be placed within a crossbar array and the logical state of the logic gate is represented as a resistance, as in a memristive memory. A MAGIC NOR gate fulfills both of these requirements. The structure of a memristive crossbar array and two-input MAGIC NOR gate within a crossbar is shown in Figure 4.

While a memristive IMPLY logic gate can also be integrated within a memristive crossbar array [11], this memristive logic family requires an additional resistor within each row of the crossbar. Additionally, unlike the NOR Boolean operation, the IMPLY operation is not logically complete and requires the operation of FALSE (writing a logical zero to a memristor). A comparison between memristive IMPLY and MAGIC is listed in Table 1.

TABLE I. COMPARISON BETWEEN IMPLY AND MAGIC

	IMPLY [11]	MAGIC
No. of voltages	2 (V_{SET}, V_{COND})	1 (V_0)
Separate input and output	No	Yes
Basic functions	IMPLY (+ FALSE)	OR, AND, NOR, NAND, NOT
No. of memristors for NOR/NAND	3 (+ a resistor)	3
No. of steps for NOR/NAND	4	2
Within memory	Yes	Yes (for NOR)
Logically complete	Requires FALSE	Yes (NOR, NAND)

V. EVALUATION AND DESIGN CONSIDERATIONS FOR A MAGIC NOR GATE

The speed of a MAGIC NOR gate is evaluated in SPICE for a 0.18 µm CMOS process. A memristor model, the VTEAM model [20], which extends the TEAM model with a threshold voltage, is used with a Biolek window function [8]. The parameters of the memristors are chosen to produce a switching time of 1 ns for a voltage pulse of 1 volt for RESET and 2 volts for SET, and to fit practical devices, as reported in [21]. The parameters of the circuit simulations are listed in Table 2.

The behavior and speed of a MAGIC NOR gate for different values of V_0 are shown in Figure 5. To evaluate the delay of the logic gate, the slowest input case is considered. The delay of a MAGIC NOR gate is determined from an input combinations of {1,0} or {0,1}.

From (1) and (2), V_0 can vary from 0.6 to 1.5 volts for the parameters listed in Table 2. As shown in Figure 5b, increasing the applied voltage V_0 decreases the delay of the logic gate. For V_0 at 1 volt, the delay of the logic gate is 1.3 ns, an increase of 30% as compared to the switching time of a single memristor.

VI. ADDITIONAL MAGIC GATES

With the same design principles described in Section II, additional Boolean functions can be provided as part of the MAGIC family. The additional MAGIC gates described in this section are not placed within a crossbar array (except for the NOT gate), but can be used as standalone logic.

TABLE II. MEMRISTOR PARAMETERS (FOR VTEAM MODEL [20])

k_{on}	-216.2 m/sec
k_{off}	0.091 m/sec
$V_{T,ON}$	-1.5 volt
$V_{T,OFF}$	0.3 volt
x_{on}	0
x_{off}	3 nm
α_{on}	4
α_{off}	4
R_{ON}	1 kΩ
R_{OFF}	300 kΩ

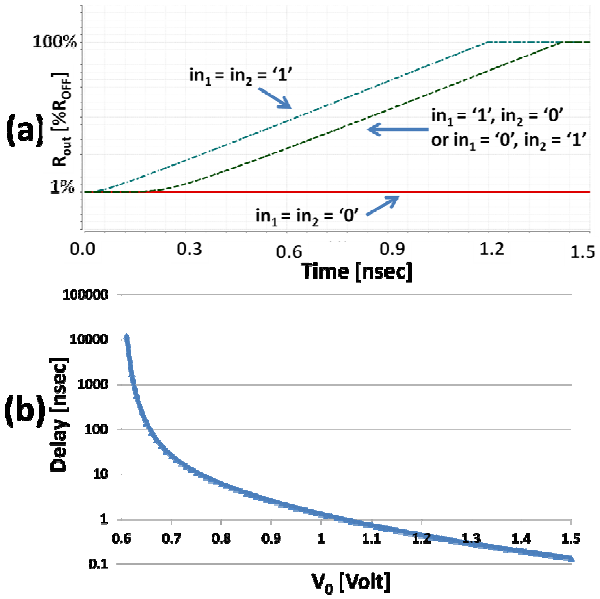


Figure 5. SPICE simulations of a two input MAGIC NOR gate. (a) Output memristor for different input combinations, V_0 = 1 volt. The delay is evaluated as the time required to switch the output memristor to logical zero when one input is logical one and the other input is logical zero (dashed line), and (b) delay for different values of voltage V_0.

Connecting the input memristors in series within the same topology as in the MAGIC NOR gate produces a NAND gate, as shown in Figure 6. OR and AND logic gates have a similar structure as, respectively, NOR and NAND, except for the opposite polarity of the output memristor *out*. Unlike NAND and NOR, *out* is initialized to logical zero prior to execution. The schematic and behavior of OR and AND MAGIC gates are shown in Figure 7. Similar to MAGIC NOR and NAND gates, multi-input logic gates are also possible for MAGIC OR and AND gates.

A MAGIC NOT gate (inverter) consists of an input memristor *in* and an output memristor *out*. The memristors are connected in series with an opposite polarity in a complementary memristor structure (or complementary resistive switches) [15], as shown in Figure 8a. In the first stage of execution, the output memristor is initialized to logical one. When applying V_0 at the gateway of the circuit, the voltage divider between *in* and *out* determines whether the resistance of the output memristor changes. For the case where *in* is logical zero, the voltage across *out* is below the threshold voltage and the logical state of *out* remains logical one, as desired. Note that in this case, the voltage at *in* is relatively

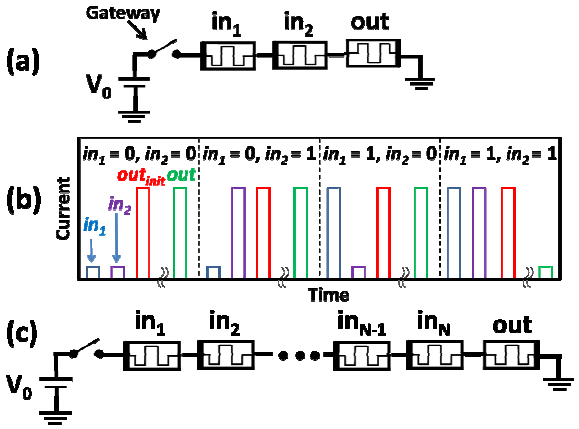


Figure 6. MAGIC NAND gate. (a) Schematic of a two input NAND gate. The logic gate consists of two input memristors *in1* and *in2*, and an output memristor *out*. During execution, a voltage V_0 is applied at the gateway of the circuit. (b) Simulation results for a two input NAND gate for all input combinations. The different curves exhibit the currents of each memristor prior to execution and after applying V_0, and (c) schematic of an N input NAND gate.

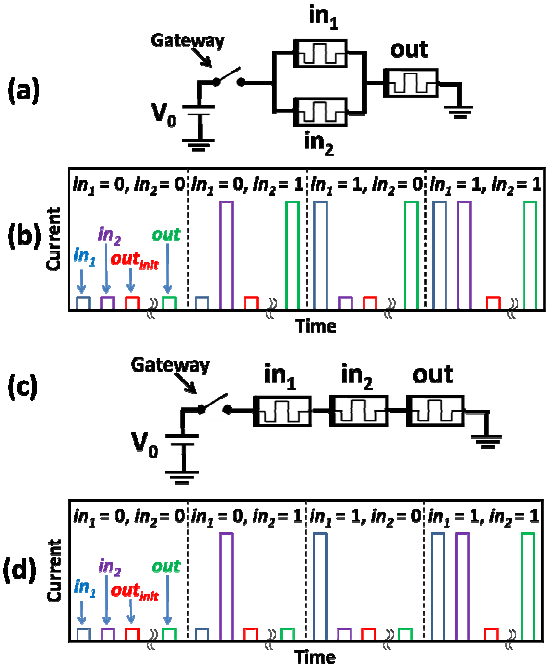


Figure 7. MAGIC OR and AND gates. The structure of the logic gates is similar to MAGIC NOR and NAND gates. The output memristor *out* is connected with the same polarity as the input memristors, and is initialized to logical zero. (a) Schematic of a two input OR gate. (b) Simulation results for a two input OR gate for all input combinations. (c) Schematic of a two input AND gate. (d) Simulation results for a two input AND gate for all input combinations.

high and the logical state at *in* therefore may be switched to logical zero. Hence, the MAGIC NOT operation can be destructive to the input unless the applied voltage at memristor *in* is below $V_{T,ON}$. For the case where *in* is logical one, the voltage across memristor *out* is sufficient to switch the logical state of *out* (greater than the threshold voltage) to logical zero. Simulation results for a NOT gate are shown in Figure 8b. A summary of several MAGIC gates, including the design constraints, is listed in Table 3.

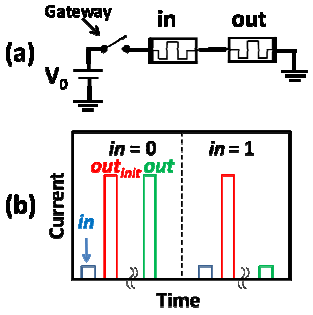


Figure 8. MAGIC NOT gate. (a) Schematic of a NOT gate. The logic gate consists of an input memristor *in* and an output memristor *out*, and (b) simulation results for a NOT gate.

VII. Conclusions

MAGIC, a novel method for memristor-based logic, is presented in this paper. Five basic logic functions, NOT, AND, NAND, NOR, and OR, use simple connections among memristors, where the number of memristors is equal to the number of inputs plus one additional memristor at the output.

Only one applied voltage controls these logic gates, different than the memristor-based IMPLY logic gate. Unlike the IMPLY gate, the input and output in MAGIC are separated, and the output is written to a dedicated memristor. The use of MAGIC NOR gates within a memristive crossbar can lead to more efficient systems in terms of performance and power consumption, and to novel non-von Neumann architectures.

References

[1]L. O. Chua, "Memristor – the Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[2]D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, Vol. 453, pp. 80-83, May 2008.

[3]Y. Ho, G. M. Huang, and P. Li, "Nonvolatile Memristor Memory: Device Characteristics and Design Implications," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 485-490, November 2009.

[4] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinalli, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Mederios-Riberio, and R. S. Williams, "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic," *Nano Letters*, Vol. 9, No. 10, pp. 3640-3645, 2009.

[5]J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable 'Stateful' Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873-876, April 2010.

[6]E. Lehtonen, J. H. Poikonen, and M. Laiho, "Two Memristors Suffice to Compute All Boolean Functions," *Electronics Letters*, Vol. 46, No. 3, pp. 239-240, February 2010.

[7]M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching Dynamics in Titanium Dioxide Memristive Devices," *Journal of Applied Physics*, Vol. 106, 074508, October 2009.

[8]S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.

[9]S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser, "Memristor-based IMPLY Logic Gate Design Procedure," *Proceedings of the IEEE International Conference on Computer Design*, pp. 142-147, October 2011.

[10]S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, " MRL – Memristor Ratioed Logic for Hybrid CMOS-Memristor Circuits," *IEEE Transactions on Nanotechnology* (in review).

[11]S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI)* (in press).

[12]S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic," *Proceedings of the International Cellular Nanoscale Networks and their Applications*, pp. 1-6, August 2012.

[13] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209-223, February 1976.

[14] L. O. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A: Materials Science & Processing*, Vol. 102, No. 4, pp. 765-783, March 2011.

[15] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary Resistive Switches for Passive Nanocrossbar Memories," *Nature Materials*, Vol. 9, No. 5, pp. 403–406, April 2010.

[16] R. Patel, S. Kvatinsky, E. G. Friedman, and A. Kolodny, "Multistate Register Based on Resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI)* (in review).

[17] E. Lehtonen and M. Laiho, "Stateful Implication Logic with Memristors," *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 33-36, July 2009.

[18] Y. Levy, J. Bruk, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaacobi, and S. Kvatinsky, "Logic Operation in Memory Using a Memristive Akers Array," *Microelectronics Journal* (in review).

[19] Y. V. Pershin and M. Di Ventra, "Practical Approach to Programmable Analog Circuits with Memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 8, pp. 1857-1864, August 2010.

[20] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: Voltage Threshold Adaptive Memristor Model," *CCIT Technical Report #856*, April 2014.

[21] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive Devices for Computing," *Nature Nanotechnology*, Vol. 8, pp. 13-24, January 2013.

Table III. Summary of MAGIC Gates

Function	Design Constraints	Design Constraints – Multiple Inputs	Within Crossbar?
NOR	$2V_{T,OFF} < V_0 < \min\left[\frac{R_{OFF}}{2R_{ON}} \cdot V_{T,OFF}, \lvert V_{T,ON}\rvert\right]$	$\frac{V_{T,OFF}}{R_{ON}} \cdot \left[R_{ON} + \left(\frac{R_{OFF}}{\chi - 1}\right) \parallel R_{ON}\right] < V_0 < \min\left[V_{T,OFF} \cdot \left(1 + \frac{R_{OFF}}{\chi R_{ON}}\right), \left(1 + \frac{\chi R_{ON}}{R_{OFF}}\right) \cdot \lvert V_{T,ON}\rvert\right]$	Yes
NAND	$3V_{T,OFF} < V_0 < \min\left[\lvert V_{T,ON}\rvert, \left(2 + \frac{R_{OFF}}{R_{ON}}\right) \cdot V_{T,OFF}\right]$	$(\chi + 1) \cdot V_{T,OFF} < V_0 < \min\left[\lvert V_{T,ON}\rvert \cdot \left(1 + \frac{\chi R_{ON}}{R_{OFF}}\right), \left(\chi + \frac{R_{OFF}}{R_{ON}}\right) \cdot V_{T,OFF}\right]$	No
OR	$V_{T,ON} < V_0 < 1.5V_{T,ON}$	$V_{T,ON} < V_0 < \left(1 + \frac{1}{\chi}\right) V_{T,ON}$	No
AND	$V_{T,ON} < V_0 < 2V_{T,ON}$	$\left(1 + \frac{R_{ON}}{R_{OFF}} \chi\right) V_{T,ON} < V_0 < \left(2 + \frac{R_{ON}}{R_{OFF}} (\chi - 1)\right) V_{T,ON}$	No
NOT	$2V_{T,OFF} < V_0 < \frac{R_{OFF}}{R_{ON}} \cdot \min\left(V_{T,OFF}, \lvert V_{T,ON}\rvert\right)$	-	Yes

MRL – Memristor Ratioed Logic

Shahar Kvatinsky, Nimrod Wald, Guy Satat,
Avinoam Kolodny, and Uri C. Weiser
Department of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 32000 ISRAEL
{skva@tx, guys@tx, kolodny@ee,
uri.weiser@ee}.technion.ac.il

Eby G. Friedman
Department of Electrical and Computer Engineering
University of Rochester
Rochester, New York 14627 USA
friedman@ece.rochester.edu

***Abstract*— Memristive devices are novel structures, developed primarily as memory. Another interesting application for memristive devices is logic circuits. In this paper, MRL (Memristor Ratioed Logic) - a hybrid CMOS-memristive logic family - is described. In this logic family, OR and AND logic gates are based on memristive devices, and CMOS inverters are added to provide a complete logic structure and signal restoration. Unlike previously published memristive-based logic families, the MRL family is compatible with standard CMOS logic. A case study of an eight-bit full adder is presented and related design considerations are discussed.**

I. INTRODUCTION

Memristors [1] and memristive devices [2] are novel structures, useful in many applications. This device is basically a resistor with varying resistance, dependent on the history of the device. It can be used for memory, where the data is stored as a resistance. While memory is the common application for memristive devices, additional applications can also use memristive devices as building blocks, such as analog circuits, neuromorphic systems, and logic circuits.

This paper is focused on bipolar memristive devices [3], such as TiO_2 memristive devices and STT-MRAM (Spin Transfer Torque Magnetoresistance Random Access Memory). In bipolar memristive devices, the resistance of the device increases due to current flow in one direction, and decreases due to current flow in the other direction. The symbol and polarity of a memristive device are shown in Figure 1. Several memristive device models have been developed. In this paper, the TEAM model [4] is used since this model can fit any memristive device.

Practical memristive devices are nonvolatile and compatible with standard CMOS technology [5]. These devices are fabricated in the metal layers of an integrated circuit, where the memristive effects occur in the oxide between the metal layers (*e.g.*, in TiO_2) or within the metal layers (*e.g.*, in STT-MRAM). Memristive devices can therefore be fabricated above the CMOS transistors. The size of a typical memristive device is relatively small, since the fabrication process is similar to the processing of a via between metal layers. Hence,

Figure 1. Memristive device symbol. The thick black line on the left side of the device represents the polarity of the device. If the current flows into the device, the resistance of the device decreases. If the current flows out of the device, the resistance increases.

memristive-based circuits may be smaller than transistor-only CMOS circuits. Memristive devices therefore exhibit high density and good scalability. The read and write time for these devices can be as fast as one nanosecond [6]. Currently, except for STT-MRAM, memristive devices suffer from endurance limitations, where the number of allowed writes per cell is approximately 10^{10} [7]. It is believed however that this limit will increase to at least 10^{15} [8]. Memristive devices may therefore solve many major problems in the semiconductor industry, providing nonvolatile, dense, fast, and power efficient memory.

Integrating memristive devices and CMOS for performing logical operations may be beneficial. Since memristive devices are fabricated within the metal layers, the integration saves physical area and therefore increases the logic density. Furthermore, with deeply scaled CMOS, CMOS logic suffers from problems such as leakage current, requiring novel logic structures.

Logic operations with memristive devices open opportunities for novel functionality. Although the use of memristive devices as logic gates is early, several approaches have been proposed, mainly for logic gates designed within the structure of a crossbar array originally targeted for memory [9]. Logic in a memristive-based crossbar opens an opportunity to explore advanced computer architectures different from the classical Von Neumann architecture. In these architectures, the memory can perform logic operations on the same devices that store data. The decision regarding which elements act as logic gates and which act as memory

This work was partially supported by Hasso Plattner Institute and by an Intel grant, "Heterogeneous Computing, the Inevitable Solution: Power Management, Scheduling and ISA," Grant no. 864-737-13.

978-1-4673-0289-0/12/$31.00 ©2012 IEEE

cells can be done dynamically during the operation of the memory [10].

Material implication (IMPLY logic gate) [11] is one option for logic inside a memristive-based crossbar. The IMPLY logic gate is extended in [12] to a NOR logic gate. Another logic family within a crossbar is MAGIC [13]. In MAGIC, all basic Boolean functions can be produced, *e.g.*, AND, NAND, NOR, and OR logic gates. All of these logic gates require a sequencer to operate the logic gate, *i.e.*, any basic Boolean function requires more than one clock cycle to execute the computation. The logic within a crossbar is therefore relatively slow. These logic gates also suffer from state drift and lack signal restoration [14].

Memristive-based logic families within a crossbar cannot be easily integrated with standard CMOS logic. In these logic families, a resistance, rather than a voltage, represents the logical state. To integrate memristive devices with CMOS for logic circuits, several requirements need to be fulfilled: the technology of the memristive devices needs to be compatible with a standard CMOS process, the logical state, used for input and output signal transfer between the logic gates, needs to be converted from a resistance into a voltage, and the interface between the memristive device layers and the CMOS layer should require minimal additional circuitry. To integrate these logic families with standard voltage based CMOS logic, a conversion mechanism is required. This mechanism includes a sense amplifier as well as additional components. The additional required circuitry reduces the efficiency of integrating CMOS and memristive-based logic families within a memory [10].

In this paper, MRL (Memristor Ratioed Logic) for integration with CMOS is described. This logic family uses the programmable resistance of memristive devices for computation of Boolean AND/OR functions with voltage as the state variable, hence it avoids the drawbacks described above. Design principles and constraints of this logic family are discussed in Section II. A case study of an eight-bit full adder is used to demonstrate the MRL design process in Section III. The dependence of the MRL gates on the behavior of the memristive device, as well as several tradeoffs in the design procedure is discussed in Sections III and IV.

II. MEMRISTOR RATIOED LOGIC (MRL) FAMILY

An interesting method for integrating memristive devices with standard CMOS logic is using memristive devices as computational elements, OR and AND logic gates [15]. Since these functions are non-inverting logic gates, a complete logic structure can be achieved by adding a standard CMOS inverter. In this logic family, the logical is represented as a voltage, consistent with CMOS. The memristive devices are utilized solely for logic computation and not for storing a logical state. The computational result is independent of the initial state of the memristive devices, and the initial state only affects the computational time. Unlike other logic methods (such as IMPLY), the computational process is composed of only a single step. Similar to standard combinatorial logic using CMOS, the topology of the circuit determines the logical function.

A. Description of Logic Gates

Both OR and AND logic gates consist of two memristive devices connected in series with opposite polarity, as shown in Figures 2a and 2b. The output node is the common node of the memristive devices, while the signals on the other terminal of each memristive device are the inputs of the logic gate.

Due to the polarity of the memristive devices, in an OR logic gate, when current flows into the logic gate through one of the inputs, the resistance of this memristive device decreases. Similarly, in an AND logic gate, the opposite polarity is used, and the resistance of the memristive device increases when current flows into the device.

Both the OR and AND logic gates react similarly to identical inputs (where either both inputs are logical 1 or both are logical 0). For identical inputs, the voltage drop between inputs is zero; hence no current flows within the circuit. The output voltage is therefore equal to the input voltage. For the case where both inputs are logical zero (one), the ground (supply) voltage is at the inputs, the output voltage is ground (supply) and the logical state of the output is logical zero (one).

For the case where the inputs are different, *i.e.*, one input is logical one and the other input is logical zero, current flows from the high voltage (the terminal of the memristive device where the input is logical one) to the low voltage (the terminal of the memristive device where the input is logical zero), thus changing the resistance of both memristive devices. This case for an OR logic gate is illustrated in Figure 2c. The resistance of the memristive device connected to the logical one input R_1 is lower, and the resistance of the memristive device R_2 is higher, as shown in Figure 2e. At the end of the computational process, the resistance of both memristive devices is approximately R_{ON} and R_{OFF}, respectively, the minimum and maximum resistance of the device. Assuming $R_{OFF} >> R_{ON}$, the output voltage of the logic gate is determined by the voltage divider across both of the memristive devices,

$$V_{out,OR} = \frac{R_{off}}{R_{off} + R_{on}} V_{high} \approx V_{high}. \quad (1)$$

In the AND logic gate, the opposite polarity, as compared to the OR logic gate, is used. For the case where the inputs are different, the resistance of the memristive devices is the opposite of the resistance of the OR logic gate. This behavior is illustrated in Figures 2d and 2f. The output voltage of the AND logic gate in this case is therefore

$$V_{out,AND} = \frac{R_{on}}{R_{off} + R_{on}} V_{high} \approx 0. \quad (2)$$

Note that the initial resistance of both memristive devices has no effect on the result of the computation. The only effect

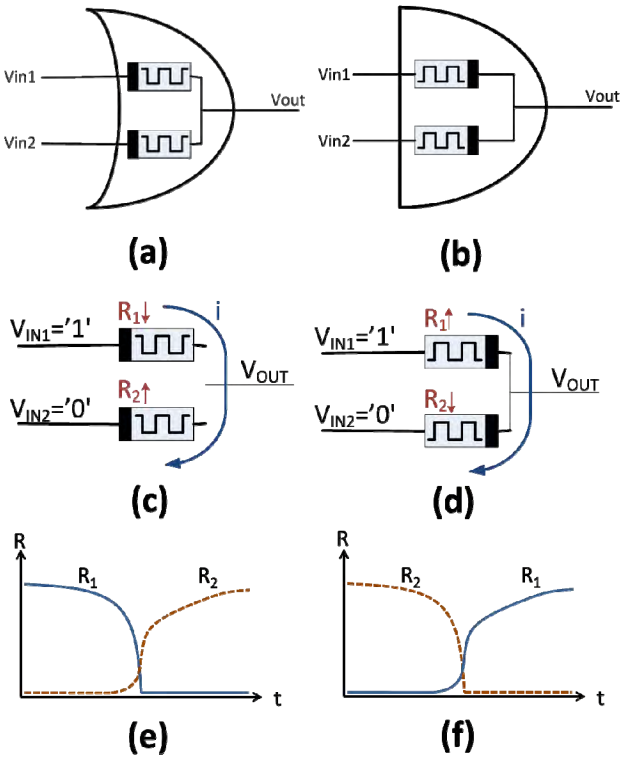


Figure 2. Schematic and behavior of MRL gates. (a) The schematic of an OR logic gate, and (b) an AND logic gate. Both logic gates consists of two memristive devices where the polarity of the memristive devices is the only structural difference. The behavior of (c) an OR logic gate, and (d) an AND logic gate when V_{IN1} = '1' and V_{IN2} = '0'. The current flows from V_{IN1} to V_{IN2} and the resistance of the memristive devices changes for the (e) OR, and (f) AND logic gates. The continuous and dashed lines are, respectively, the resistance of R_1 and R_2.

of the initial resistance on the behavior of the logic gate is the delay time of the execution for the case where the inputs are different, *i.e.*, the time required to change the resistance of both memristive devices to either the maximum or minimum resistance. The delay time is also dependent on the voltage level. A relatively low voltage level increases the delay time. It is possible that the memristive devices do not fully switch and achieve the maximum and minimum resistance since the input voltages are not applied for a sufficiently long time or the input voltage is too low. In this case, it would be difficult to distinguish between the different output levels. The MRL family is inspired by Diode Logic [16] and shares some characteristics, such as both logic families are non-inverting and non-restoring [17]. The number of inputs for both MRL gates can be extended in a similar way as diode logic, as shown in Figures 3a and 3b.

To provide a complete logic family, an inverter is needed in addition to OR and AND logic gates. Furthermore, memristive devices are passive elements and therefore cannot amplify signals. The MRL OR and AND logic gates therefore lack signal restoration, *i.e.*, the output voltage levels degrade, as expressed by (1) and (2). These logic gates cannot therefore be cascaded for too many stages before signal amplification is required. CMOS logic, alternatively, exhibits signal restoration. Since the logical state of the input and output in MRL OR and AND logic gates is represented as a voltage, these logic gates can be integrated with standard CMOS inverters. To provide a complete logic structure and signal restoration, the addition of a CMOS inverter to the MRL family is therefore proposed. The schematic of a two input MRL NAND and NOR is shown in Figures 3c and 3d.

B. General design considerations

In the design process of an MRL gate, several issues need to be considered. When the input changes from one input case to another input case, *i.e.*, changing the inputs from (0,1) to (1,0) and vice versa, the output produces a dynamic hazard until the switching process is completed. Another issue may occur when both initial resistances are high (approximately R_{OFF}). In this case, the current through the logic gate is relatively small, and the settling time is therefore relatively long, also producing a dynamic hazard. The dynamic behavior of the OR and AND logic gates is illustrated in Figures 4a and 4b.

Power consumption is another issue. When both inputs are identical, no current flows in the circuit and the power is zero. If the inputs are different, current flows and power is consumed. The power consumed during the switching of the memristive devices is dependent on the resistance of both memristive devices and changes during the computational process. Generally, the power consumption of an MRL gate for these input cases is

$$P(t) = \frac{V_{high}^{\ 2}}{R_1(t) + R_2(t)}, \tag{3}$$

where V_{high} is the voltage of logical one and is assumed to be constant, and $R_1(t)$ and $R_2(t)$ are the resistance of the memristive devices, which change during the computational process. The value of $R_1(t)$ and $R_2(t)$ is dependent on the initial states and the value of V_{high}. For the case of different inputs, a constant current flows from one input to the other input, even after the resistance of the memristive devices reaches the desired magnitude and the output becomes stable. The static power consumed in these cases is approximately

$$P_{static} = \frac{V_{high}^{\ 2}}{R_{on} + R_{off}}. \tag{4}$$

The power consumption for all input cases is illustrated in Figure 4c. The output voltage is dependent on the voltage divider across the two memristive devices. This voltage divider degrades the output signal. Although the degradation is minor when $R_{OFF} >> R_{ON}$, for cascaded logic gates, this degradation accumulates and may become significant. This phenomenon can be avoided by occasionally amplifying the signal by CMOS inverters or buffers. Integrating a CMOS inverter into an MRL OR or AND logic gate however adds capacitance to the circuit. The delay time of the logic gates is dependent on the CMOS gate capacitance and therefore needs to be optimized. The delay of the logic gates is the time required for the memristive devices to be fully switched, and

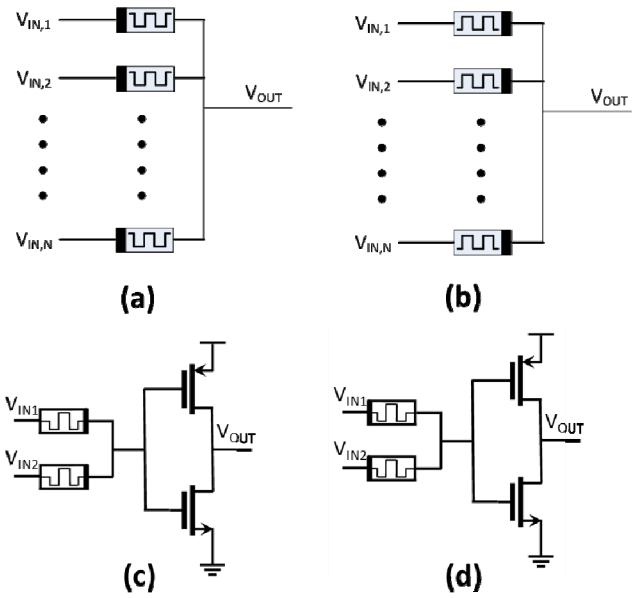


Figure 3. Schematic of an (a) N-input MRL OR, (b) N-input MRL AND, (c) two-input MRL NAND, and (d) two-input MRL NOR.

is dependent on the determined by the case of different inputs.

The MRL logic gates can be inserted into a standard cell library as in standard CMOS logic. These standard cell libraries can consist of OR and AND logic gates. Alternatively, NOR and NAND logic gates, consisting of a memristive-based OR (AND), and a CMOS inverter, can produce the functionality of a NOR (NAND) logic gate.

III. EIGHT-BIT FULL ADDER CASE STUDY

An eight-bit full adder is considered as a case study for the MRL family. Five different parameter sets of memristive devices are chosen to evaluate a variety of memristive devices. The primary parameters are the linearity coefficient and the current threshold (respectively, α, i_{on}, and i_{off} in the TEAM model [4]). All other parameters are chosen to exhibit a hysteretic behavior. The parameters for the memristive devices are listed in Table 1.

To provide a standard cell design methodology, the standard cell is a NAND (NOR) logic gate, as described in Section IIB. No current flows from the output node in steady state since the output node of the AND (OR) logic gate is connected to an MOS gate. In this approach, every standard cell requires two connections between the CMOS and memristive layers, one for the middle stage transition and one for the output. This approach is robust, albeit inefficient in terms of power consumption and area as compared to an optimized circuit, where the CMOS inverter is only applied when signal restoration is needed or when the logic function requires signal inversion. In this case study, the optimized approach is used.

For the optimized approach, when connecting cascaded memristive-based MRL gates, current can flow from the output node into the input of the next logic gate. In this case, the current flowing through the two memristive devices of one gate is not equal, and the smaller current may drop below the current threshold of the memristive devices, causing the logic gate to partially switch. This phenomenon can degrade the output voltage, and may perhaps cause the logic to fail after a single logic stage.

TABLE 1. DIFFERENT PARAMETERS OF THE MEMRISTIVE DEVICES USED IN THE CASE STUDY

Device / Parameter	Linear with no current thresh-old	Linear with current thresh-old	Low non-linearity	Non-linear	Highly non-linear
Parameter set number	1	2	3	4	5
α	1	1	3	5	10
i_{on}	-100 fA	-20 µA	-5 µA	-5 µA	-10 µA
i_{off}	100 fA	20 µA	5 µA	5 µA	10 µA
k_{on}	$-5\cdot 10^{-8}$	-10	-0.1	-0.01	-0.001
k_{off}	$5\cdot 10^{-8}$	10	0.1	0.01	0.001
R_{on}	1 kΩ				
R_{off}	100 kΩ				

One approach to eliminate a possible logic failure is to increase the voltage of the high logical state to ensure that all currents in the circuit are greater than the current threshold of the devices. The increase in voltage is limited by the CMOS process, since high voltages may cause breakdown in the CMOS transistors (*e.g.*, gate induced drain leakage [18]), and also dissipate more power.

Another approach to eliminate logic failure is to amplify the signal with CMOS logic gates, preventing steady state current leakage and performing signal restoration. In this case study, both approaches are used. The voltage is increased and signal restoration is achieved through a CMOS inverter. The behavior of an MRL XOR logic gate is shown in Figure 5 to demonstrate the signal degradation. Note that these signal degradation issues are circuit dependent, *i.e.*, the degree of signal degradation is dependent on the logic circuit structure as well as the parameters of the memristive devices. A schematic of the one-bit full adder used in this case study is shown in Figure 6.

The design of the eight-bit full adder in this case study is achieved using eight cascaded one-bit full adders. A tradeoff between signal integrity and minimizing the number of vias is the primary issue. To maintain a distinct value for the output of the eight-bit full adder (S_i for i = 1, …, 8 and C_{OUT}), a set of CMOS buffers is added to the circuit to amplify the output signal. For the intermediate signals ($C_{OUT} \rightarrow C_{IN}$), no constraint is placed on the strength of the signal other than to maintain the correct logical polarity. A lower signal strength requires fewer CMOS gates and hence less area and power consumption. The required number of CMOS buffers is dependent on the signal degradation along the logic path.

For parameter sets 1, 3 and 4 (memristive devices with a relatively low current threshold), the one-bit full adder shown

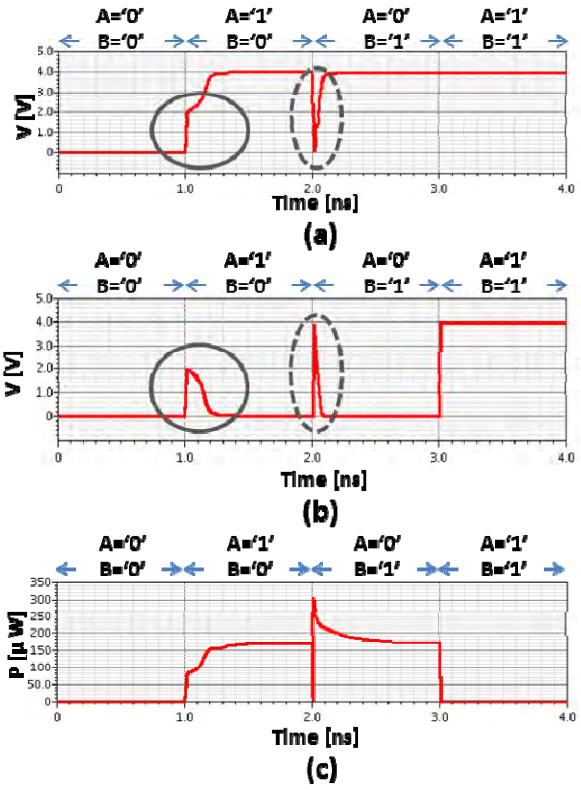


Figure 4. Dynamic behavior of MRL gates. Waveforms of (a) an OR logic gate, and (b) an AND logic gate. The output voltage is shown for different input states. Dynamic hazards occur when the input changes to ('0', '1') or ('1', '0'), which is marked by an oval. (c) The power consumption for both logic gates is identical. For the cases where the input states are different ('0', '1') or ('1', '0'), static power is consumed after the output is stable.

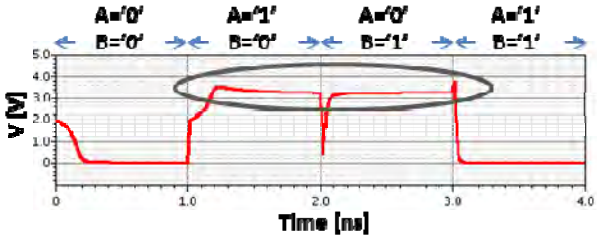


Figure 5. Dynamic behavior of an MRL XOR logic gate. The high voltage is 4 Volts. The output voltage degrades by approximately 15% for the input cases of ('1', '0') and ('0', '1').

in Figure 6 exhibits correct logic functionality, which requires amplifying the signal between different bit levels. Parameter sets 2 and 5 demonstrate a high current threshold and are therefore more sensitive to signal degradation due to partial switching. For these parameter sets, the circuit fails for all CMOS compatible voltages. For parameter sets 2 and 5, buffers have been added to the one-bit full adder circuit to ensure correct logic behavior. The required voltage levels and number of buffers for each parameter set are listed in Table 2, total number of devices is listed in Table 3, and normalized power consumption[1] for each parameter set is listed in Table 4.

[1] The power is normalized since the parameter set of the memristive devices is not correlated to the CMOS process.

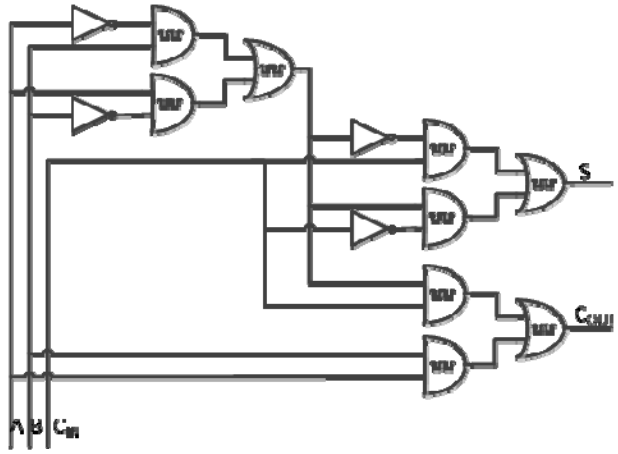


Figure 6. Schematic of an MRL one bit full adder (S = XOR[A, B, C_{IN}], $C_{OUT} = A \cdot B + C_{IN} \cdot$XOR[$A$, B]) for the optimized method used in the case study. The one-bit full adder consists of six memristive-based OR logic gates, three memristive-based AND logic gates, and four CMOS inverters. In this circuit, 18 memristive devices and eight CMOS transistors are used.

Note from the data listed in Tables 3 and 4 that unlike most digital applications [4], a linear memristive device with no threshold (as in parameter set no. 1) is preferable to minimize the number of connections between the CMOS and memristive layers, and to reduce power. The optimized approach consumes less dynamic power but more total energy, as compared to a standard cell methodology, since the static power is non-zero. Since decreasing the operating voltage requires additional CMOS buffers, the number of CMOS buffers in parameter set no. 3 (a high voltage of 3 Volts) is lower than in parameter set no. 1. The high voltage used in parameter sets number 2 and 5 significantly increases the power consumption.

IV. CONCLUSIONS

Memristor Ratioed Logic (MRL), a hybrid CMOS-memristive logic family, is described in this paper. This logic family uses less die area as compared to CMOS logic. It is possible to reduce the design effort of an MRL circuit by using standard library cells composed of only NOR and NAND logic gates. Standard cells however limit the flexibility of the design process and restrict the opportunity to save area. Other optimization criteria are also possible, such as increasing the operating voltage and minimizing the number of connections between the CMOS and memristive layers.

An eight-bit full adder is presented as a case study. This full adder is optimized for minimum CMOS/memristive connections and saves approximately 50% in area as compared to CMOS logic, while requiring 44% fewer connections and 30% less power as compared to an MRL standard cell library.

It is also shown that a linear memristive device with no current threshold is preferable for the MRL logic family, unlike other digital applications, where a threshold and nonlinearity are desirable. MRL gates based on linear memristive devices are faster, smaller, and consume less power as compared to nonlinear memristive devices.

The Memristor Ratioed Logic family opens an opportunity for additional memristive/CMOS integrated circuits and increases logic density. This enhancement can provide greater computational abilities to processor and other computational circuits.

TABLE 2. VOLTAGE LEVEL AND NUMBER OF BUFFERS FOR EACH PARAMETER SET IN THE CASE STUDY

Parameter set	Supply voltage	Number of buffers needed			
		Inside each one bit full adder	Between each C_{OUT} and C_{IN}	After last stage C_{OUT}	After each S_i
1	1V	0	2	2	1
2	6.5V	2	1	2	2
3	3V	0	1	1	1
4	4V	0	2	2	1
5	6.5V	2	2	2	2

TABLE 3. SUMMARY OF CASE STUDY

Parameter set	Number of memristors	Number of CMOS transistors	Number of vias	Supply voltage
CMOS – based	-	288	-	1 V
Standard cell approach	144	144	144	1 V
1	144	160	80	1 V
2	144	228	96	6.5 V
3	144	128	80	3 V
4	144	160	80	4 V
5	144	256	96	6.5 V

TABLE 4. POWER CONSUMPTION AND ENERGY FOR CASE STUDY

Parameter set	Average power [normalized]	Total energy [normalized]
Standard cell approach (for parameter set 1)	1	1
1	0.72	5.02
2	386.4	2035.1
3	22.5	167.9
4	60.8	499.2
5	354.95	2004.5

REFERENCES

[1] L. O. Chua, "Memristor – The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[2] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209- 223, February 1976.

[3] P. Vontobel, W. Robinett, J. Straznicky, P. J. Kuekes, and R. S. Williams, "Writing to and Reading from a Nano-Scale Crossbar Memory Based on Memristors," *Nanotechnology*, Vol. 20, No. 42, pp. 1-21, October 2009.

[4] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, April 2012 (in press).

[5] J. Borghetti, Z. Li, J. Strasnicky, X. Li, D. A. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A Hybrid Nanomemristor/Transistor Logic Circuit Capable of Self-Programming," *Proceedings of the National Academy of Sciences*, Vol. 106, No. 6, pp. 1699-1703, February, 2009.

[6] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-Nanosecond Switching of a Tantalum Oxide Memristor," *Nanotechnology*, Vol. 22, No. 48, pp. 1-7, November 2011.

[7] J. J. Yang *et al*, "High Switching Endurance in TaOx Memristive Devices," *Applied Physics Letters*, Vol. 97, No. 23, pp. 1-3, December 2010.

[8] J. Nickel, "Memristor Materials Engineering: From Flash Replacement towards a Universal Memory," *Proceedings of the IEEE IEDM Advanced Memory Technology Workshop*, December 2011.

[9] G. Snider, "Computing with Hysteretic Resistor Crossbars," *Applied Physics A*, Vol. 80, No. 6, pp. 1165-1172, March 2005.

[10] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Design Principles and Methodologies for Integrated Memristor Memory and Memristor Logic," unpublished.

[11] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable 'Stateful' Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873-876, April 2010.

[12] S. Shin, K. Kim, and S.-M. Kang, "Reconfigurable Stateful NOR Gate for Large-Scale Logic-Array Integrations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 58, No. 7, pp. 442-446, July 2011.

[13] S. Kvatinsky *et al*, "MAGIC – Memristor Aided LoGIC," unpublished.

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY Logic Design Procedure," *Proceedings of the IEEE International Conference on Computer Design*, pp. 142-147, October 2011.

[15] K. Eshraghian, course notes on "Memristive Circuits and Systems," Technion, June 2011.

[16] R. H. Wilkinson, "A Method of Generating Functions of Several Variables Using Analog Diode Logic," *IEEE Transactions on Electronic Computers*, Vol. EC-12, No. 2, pp. 112-129, April 1963.

[17] G. G. Langdon Jr., *Logic Design – A Review of Theory and Practice*, Academic Press, 1974.

[18] T. Y. Chan, J. Chen, P. K. Ko, and C. Hu, "The Impact of Gate-Induced Drain Leakage Current on MOSFET Scaling," *Proceedings of the IEEE International Electron Devices Meeting*, pp. 718-721, December 1987.

MRL - Memristor Ratioed Logic for Hybrid CMOS-Memristor Circuits

Shahar Kvatinsky, *Student Member, IEEE,* Nimrod Wald, Guy Satat, Eby G. Friedman, *Fellow, IEEE,* Avinoam Kolodny, *Senior Member, IEEE,* and Uri C. Weiser, *Fellow, IEEE*

***Abstract* — An attractive "beyond Moore" approach is to combine standard CMOS and memristors, novel devices developed primarily as memory, to perform logical operations. In this paper, MRL (Memristor Ratioed Logic) - a hybrid CMOS-memristor logic family - is described. In this logic family, OR and AND logic gates are based on memristive devices, and CMOS inverters are added to provide a complete logic structure and signal restoration. The MRL family is compatible with standard CMOS logic since the logical state is represented by voltage as in CMOS. Design issues and considerations are discussed, including area, power, and speed, and a case study of an eight-bit full adder is presented.**

***Index Terms*—Memristive systems, memristor, SPICE, logic design.**

I. Introduction

The advancements of computer capabilities over the past several decades are closely linked to the efficient exploitation of semiconductor technology. Since the 1960's, integrated circuits have provided significant growth in the number of processing elements and memory bits available to system developers. This growth has enabled increasingly complex computer hardware. Simultaneously, semiconductor technology has provided orders of magnitude improvements in speed, power consumption, and reliability, together with significant reductions in the cost per device. These trends are direct consequences of frequent miniaturization of device dimensions in the semiconductor fabrication process. The exponential rate of microelectronic device scaling cannot be sustained indefinitely. There is broad agreement that nanoscale CMOS transistor sizes will approach fundamental physical limits within the next decade [1].

Once devices can no longer be scaled, microelectronic technology will require innovations to enable continued growth in the complexity of hardware systems. These enhancements may include revolutionary new devices such as carbon nanotubes or spintronic devices rather than CMOS. Less radical hybrid approaches, combining standard CMOS with new technologies, may provide more practical and continuous growth paths during the next 20 to 30 years. An example is the fabrication of multi-layered integrated circuits (*i.e.*, three-dimensional circuits [2]) which are becoming commercially available. Other attractive new technologies which will extend the capabilities of CMOS are memristors [3] and memristive devices [4]. These devices are added in the metal layers above the standard CMOS layers, providing a significant increase in functional density.

In this paper, a novel logic family named MRL (Memristor Ratioed Logic) combining memristors with CMOS is described. This logic family may be useful to further extend CMOS technology and perform logic in the "beyond Moore" era. The rest of the paper is organized as follows: memristors and memristor-based logic circuits are described in Section II. Design principles and constraints of this logic family are discussed in Section III. A case study of an eight-bit full adder is used to demonstrate the MRL design process in Section IV. The dependence of MRL gates on the behavior of memristive devices and other design considerations are discussed in Sections IV and V.

II. Memristor-based Logic

In recent years, novel memory technologies have emerged, especially to replace Flash technology. These emerging technologies are based on two terminal devices with varying resistance and include Phase Change Memory (PCM), Resistive RAM (RRAM), and Spin Torque Transfer Magnetoresistance Random Access Memory (STT-MRAM). These technologies can be described as memristive devices [4], [15], which their existence was suggested by Chua and Kang in 1971 and 1976. The focus of this paper is on bipolar memristive devices [5], such as TiO_2 resistive switch and STT-MRAM. In bipolar memristive devices, the resistance of the device increases when current flows in one direction, and decreases when current flows in the other direction. The symbol, polarity, and behavior of a bipolar memristive device are shown in Figure 1.

Although a memristive device is typically regarded as a memory device, where the resistance represents the stored state variable, a memristor can also be used to provide

Manuscript received 13rd October, 2013. This work was partially supported by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, and by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

S. Kvatinsky, G. Satat, N. Wald, A. Kolodny, and U. C. Weiser are with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel. (S. Kvatinsky corresponding author phone: 972-77-887-1923; fax: 972-4829-5757; e-mail: skva@tx.technion.ac.il).

E. G. Friedman is with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627, USA.

combinational Boolean functions. Several memristive device models have been developed. The TEAM (ThrEshold Adaptive Memristor) model [6] is used in this paper since this model can support any memristive device. The TEAM model describes the behavior of the device by the following equations

$$\frac{dx(t)}{dt} = \begin{cases} k_{off} \cdot \left(\frac{i(t)}{i_{off}} - 1 \right)^{\alpha_{off}} \cdot f_{off}(x), & 0 < i_{off} < i, \quad (1a) \\ 0, & i_{on} < i < i_{off}, \quad (1b) \\ k_{on} \cdot \left(\frac{i(t)}{i_{on}} - 1 \right)^{\alpha_{on}} \cdot f_{on}(x), & i < i_{on} < 0, \quad (1c) \end{cases}$$

$$v(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{x_{off} - x_{on}} \left(x - x_{on} \right) \right] \cdot i(t), \quad (2)$$

where k_{off} and k_{on} are fitting parameters, α_{on} and α_{off} are the adaptive nonlinearity parameters, i_{off} and i_{on} are the current threshold parameters, $f_{on}(x)$ and $f_{off}(x)$ are window functions, R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, and x_{on} and x_{off} are, respectively, the minimum and maximum allowed value of the internal state variable x.

Practical bipolar memristive devices are compatible with standard CMOS technology [7]. These devices are fabricated in the metal layers of an integrated circuit, where the memristive effects occur in the oxide between the metal layers (*e.g.*, in TiO_2) or within the metal layers (*e.g.*, in STT-MRAM). Memristive devices can therefore be fabricated physically above the CMOS transistors. The size of a typical memristive device is relatively small, since the fabrication process is similar to processing a via between metal layers (a typical area size for a RRAM cell can be less than 4 F^2, where F is the feature size). Hence, memristive-based circuits may be more dense than transistor-only CMOS circuits. Memristive devices therefore exhibit high density and good scalability. The read and write time for these devices can be as fast as 100 picosecond [8], although currently typical values are more than a nanosecond. Currently, except for STT-MRAM, memristive devices suffer from endurance limitations, where the number of allowed writes per cell is approximately 10^{10} [9]. It is believed however that this limit will increase to at least 10^{15} [10]. In this paper, it is therefore assumed that endurance is not a critical limitation.

With deeply scaled CMOS, logic circuits suffer from problems such as leakage current, requiring novel logic structures. The use of memristors as logic circuits can save physical area, increasing logical density, thereby opening opportunities for novel functionality. Although the concept of memristive devices as logic gates is at an early stage of development, several approaches have been proposed, mainly for logic gates within a crossbar array [11], [12], [16].

Memristive-based logic families within a crossbar cannot be easily integrated with standard CMOS logic since a resistance, rather than a voltage, represents the logical state. To integrate these logic families with standard voltage based CMOS logic, a conversion mechanism is required. This added circuitry reduces the efficiency of integrating CMOS and memristive-based logic families within a memory. Furthermore, the operation of these memristor-based logic families requires a sequence of several time phases. These requirements limit the speed of the logic operation.

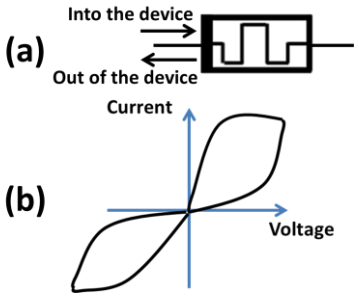


Figure 1. (a) Memristive device symbol. The thick black line on the left side of the device represents the polarity of the device. If the current flows into the device, the resistance of the device decreases. If the current flows out of the device, the resistance increases. (b) The current-voltage curve of a memristive device exhibit hysteresis for a cyclic input.

III. Memristor Ratioed Logic (MRL) Family

In this section, the MRL (Memristor Ratioed Logic) family is presented. In MRL, the programmable resistance of the memristive devices is used to compute Boolean AND/OR functions. The memristive devices are used solely as computational elements and not as memory elements. In this proposed logic family, the logical value is represented by a voltage, consistent with CMOS, enabling the integration of this hybrid logic family with standard CMOS logic. Since OR and AND functions are non-inverting, a complete logic structure can be achieved by adding a standard CMOS inverter.

The MRL family is different from other hybrid CMOS-memristor circuits, where the memristors act as configurable switches. In previous hybrid CMOS-memristor circuits, the resistance of the memristive devices is programmed prior to operation, and does not change during execution (see *e.g.*, [7], [13], [14]).

A. Memristor-based logic gates

Two-input OR and AND logic gates consist of two memristive devices connected at opposite polarities, as shown in Figures 2a and 2b. The output node is the common node of the memristive devices, while the signal on the other terminal of each memristive device is an input to the logic gate.

Due to the polarity of the memristive devices, in an OR logic gate, when current flows into the logic gate through one of the inputs, the resistance of the memristive device decreases. Similarly, in an AND logic gate, the opposite polarity is used, and the resistance of the memristive device increases when current flows into the device.

For identical inputs, the voltage drop between the inputs is zero; hence no current flows within the circuit. The output voltage is therefore equal to the input voltage. For the case where both inputs are logical zero (one), the ground (supply)

voltage is at the inputs, the output voltage is ground (supply), and the logical state of the output is logical zero (one).

For the case where the inputs are different, *i.e.*, one input is logical one and the other input is logical zero, current flows from the high voltage (the terminal of the memristive device where the input is logical one) to the low voltage (the terminal of the memristive device where the input is logical zero), thereby changing the resistance of both memristive devices. An OR logic gate is illustrated in Figure 2c. The resistance of the memristive device connected to the logical one input R_1 becomes lower, and the resistance of the memristive device R_2 becomes higher, as shown in Figure 2e. At the end of the computational process, the resistance of both memristive devices is approximately R_{ON} and R_{OFF}, respectively, the minimum and maximum resistance of the device. Assuming $R_{OFF} >> R_{ON}$, the output voltage of the logic gate is determined by the voltage divider across the memristive devices,

$$V_{out,OR} = \frac{R_{off}}{R_{off} + R_{on}} V_{high} \approx V_{high}. \quad (1)$$

In the AND logic gate, the opposite polarity as compared to the OR logic gate is used. The behavior is illustrated in Figures 2d and 2f. The output voltage of the AND logic gate in this case is

$$V_{out,AND} = \frac{R_{on}}{R_{off} + R_{on}} V_{high} \approx 0. \quad (2)$$

Note that the initial resistance of both memristive devices has no effect on the result of the computation. The only effect of the initial resistance on the behavior of the logic gate is the delay of the execution for the case where the inputs are different, *i.e.*, the time required to change the resistance of both memristive devices to either the maximum or minimum resistance. The delay is also dependent on the voltage level of the input signal. A relatively low voltage level increases the delay. It is possible that the memristive devices do not fully switch and achieve the maximum and minimum resistance since the input voltages are not applied for a sufficiently long time or the input voltage is too low. In this case, it would be difficult to distinguish between the different output levels. The MRL family is inspired by Diode Logic and shares some characteristics, as both logic families are non-inverting and non-restoring. The number of inputs for both MRL gates can be extended in a similar way as diode logic, as shown in Figure 3a and 3b. MRL gates can also be cascaded as shown in Figure 3e.

B. Adding CMOS-based inversion

To provide a complete logic family, an inverter is needed in addition to the OR and AND logic gates. Furthermore, memristive devices are passive elements and therefore cannot amplify signals. The MRL OR and AND logic gates lack signal restoration, *i.e.*, the output voltage levels degrade, as expressed by (1) and (2). These logic gates cannot be cascaded for too many stages before signal amplification is required.

Since the logical state of the input and output in MRL logic gates is represented as a voltage, these logic gates can operate with standard CMOS inverters. The addition of a CMOS inverter to the MRL family solves both problems. A schematic of a two input MRL NAND and NOR is shown in Figures 3c and 3d.

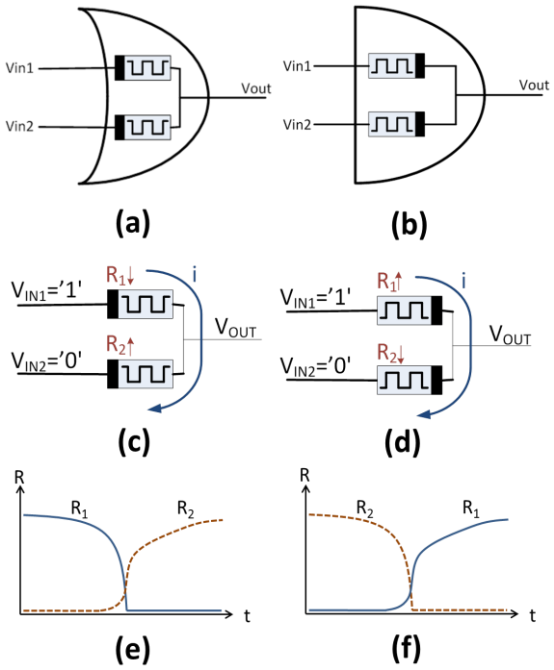


Figure 2. Schematic and behavior of MRL gates. (a) The schematic of an OR logic gate, and (b) an AND logic gate. Both logic gates consist of two memristive devices where the polarity of the memristive devices is the only structural difference. The behavior of (c) an OR logic gate, and (d) an AND logic gate when V_{IN1} = '1' and V_{IN2} = '0'. The current flows from V_{IN1} to V_{IN2} and the resistance of the memristive devices changes for the (e) OR, and (f) AND logic gates. The continuous and dashed lines are, respectively, the resistance of R_1 and R_2.

C. Dynamic behavior and speed of a single logic gate

In the design process of an MRL gate, several issues need to be considered. These characteristics include the delay of the logic gate, power dissipation, and output signal degradation.

The speed of the logic gate is determined by the time required to achieve the desired resistance when the inputs are different (logical one and zero). The delay needs to be determined for the case where both memristors have an initial resistance of R_{OFF} and the initial current is therefore minimal. In this case, the delay is the time required to change the resistance of one of the memristors to R_{ON}. This time depends on the supply voltage and the specific behavior of the memristors. The delay as a function of supply voltage is shown in Figure 4a. The delay can be approximately determined from

$$T = \frac{kQ' R_{OFF}}{V_{high}}, \quad (3)$$

where Q' is the charge required to switch the resistance and V_{high} is the logical one voltage. k is a constant that depends upon the memristor model. For the binary resistance model [12], k equals 2, and for the linear ion drift model [6], k equals 1.5. Note that a lower R_{OFF} or a higher voltage decreases the delay.

The switching process may also produce dynamic hazards. When the input changes from one input case to another input

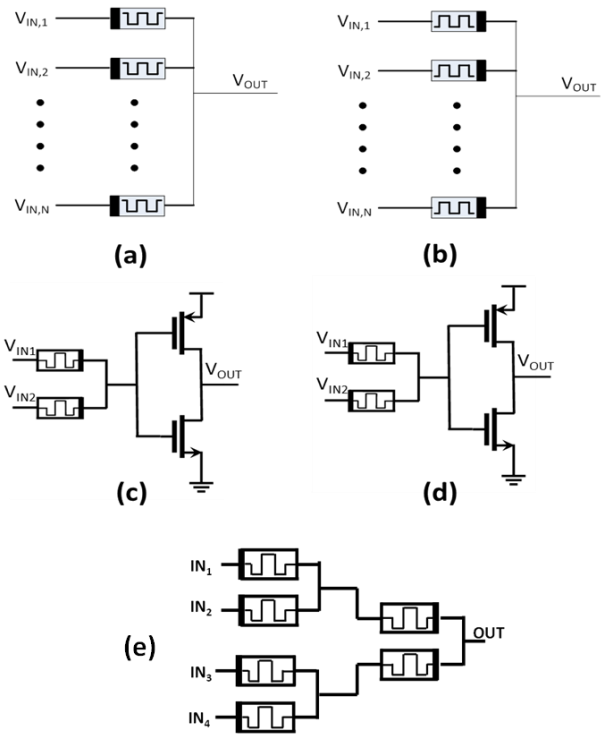


Figure 3. Schematic of an (a) N-input MRL OR, (b) N-input MRL AND, (c) two-input MRL NAND, (d) two-input MRL NOR, and (e) a two-stages MRL gates, where the first stage consists of OR gates and the second stage is an AND gate.

case, *i.e.*, changing the inputs from (0,1) to (1,0) and vice versa, the output produces a dynamic hazard until the switching process is completed. The dynamic behavior of the OR and AND logic gates for different input cases is illustrated in Figures 5a and 5b. During the switching process, the output value is erroneous, although this behavior occurs for a shorter period than the delay of the logic gate.

D. Power consumption of a single logic gate

When both inputs are identical, no current flows in the circuit and the power is zero. If the inputs are different, current flows and power is consumed. The power to switch the memristive devices depends upon the resistance of both memristive devices and changes during the computational process. Generally, the power consumption of an MRL gate for these input cases is

$$P(t) = \frac{V_{high}^{\ 2}}{R_1(t) + R_2(t)}, \tag{4}$$

where V_{high} is the voltage of logical one (supply voltage) and is assumed to be constant, and $R_1(t)$ and $R_2(t)$ are the resistance of the memristive devices, which changes during the computational process. The value of $R_1(t)$ and $R_2(t)$ depends upon the initial states and the value of V_{high}. For the case of different inputs, a constant current flows from one input to the other input after the resistance of the memristive devices reaches the desired magnitude and the output becomes stable as illustrated in Figure 5c. The static power consumed in these cases is approximately

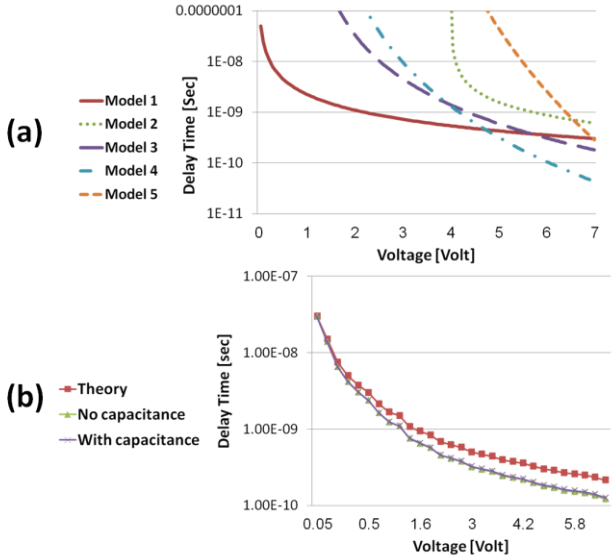


Figure 4. Delay of an MRL gate as a function of supply voltage. (a) Delay for different memristor characteristics as listed in Table 1 and (b) the delay for linear ion drift memristor [6]. The difference in the delay of an MRL gate with and without a load capacitance of 10 fF is relatively small (maximum difference of 4%). The difference between the theoretical and simulated delay increases with the supply voltage (varies from 1% to 40%). The parameters for the linear ion drift model are $\mu_V = 10^{-5}$ m^{2}s^{-1}V^{-1}, $R_{OFF} = 100$ kΩ, $R_{ON} = 1$ kΩ, $D = 10$ nm ($Q' = 10^{-14}$ C).

$$P_{static} = \frac{V_{high}^{\ 2}}{R_{on} + R_{off}}. \tag{5}$$

The static power is a disadvantage as compared to CMOS logic. To lower the static power, the input signals need to be removed once the output state becomes stable. There is a need to activate the circuit only for the time required to execute the computation (*i.e.*, the delay of the logic gates) and store the result. It is desired therefore to pipeline the execution.

E. Concatenating MRL gates

The output voltage is dependent on the voltage divider across the two memristive devices. This voltage divider degrades the output signal. For memristors with a current threshold [6], the tolerable degradation is limited by this threshold. Greater degradation can cause incorrect operation. Although the degradation for a single MRL gate is relatively small when $R_{OFF} >> R_{ON}$, for cascaded logic gates, this degradation accumulates and may become significant.

The degradation as a function of the number of logical stages and the R_{OFF}/R_{ON} ratio is shown in Figure 6. The behavior of an MRL XOR logic gate is depicted in Figure 7 to illustrate the signal degradation. Note that these signal degradation issues are circuit dependent, *i.e.*, the degree of signal degradation is dependent on the structure of the logic circuit as well as the specific parameters of the memristive devices.

The degradation phenomenon can be avoided by occasionally amplifying the signal by CMOS inverters or buffers. Integrating a CMOS inverter into an MRL OR or AND logic gate however adds capacitance to the circuit. The delay of the logic gates is therefore also dependent on the

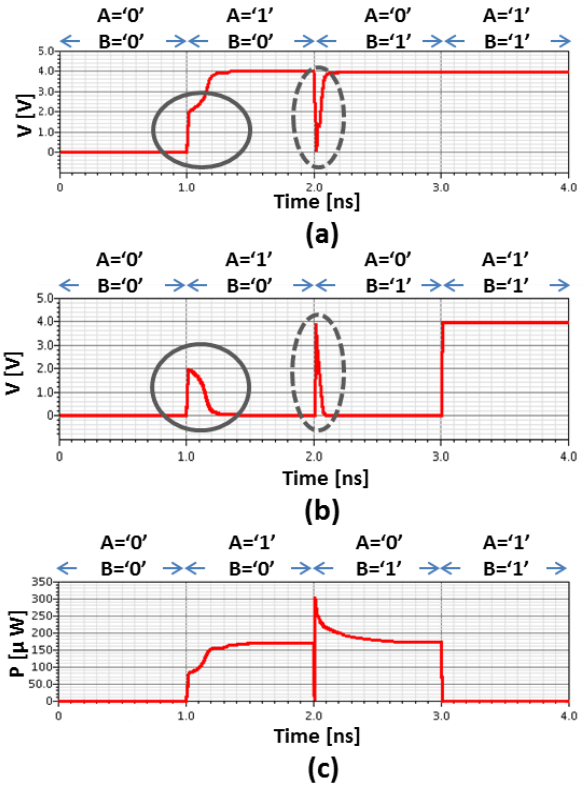


Figure 5. Dynamic behavior of MRL gates. Waveforms of (a) an OR logic gate, and (b) an AND logic gate. The output voltage is shown for different input states. Dynamic hazards occur when the input changes to ('0', '1') or ('1', '0'), which is marked by an oval. (c) The power consumption for both logic gates is identical. For the cases where the input states are different ('0', '1') or ('1', '0'), static power is consumed after the output is stable.

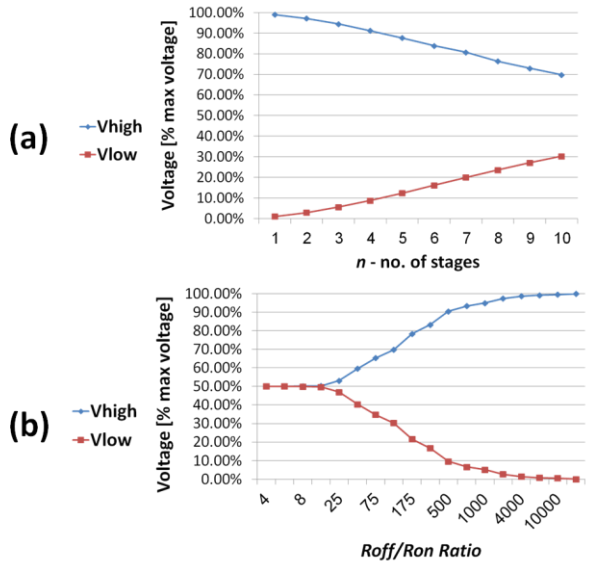


Figure 6. Output degradation. (a) Output degradation as a function of the number of logical stages. R_{OFF}/R_{ON} = 100. The maximum degradation for five and ten logical stages is, respectively, 12.4% and 30.25%. (b) Output degradation as a function of the R_{OFF}/R_{ON} ratio for ten logical stages. The maximum degradation for a ratio of 100, 500, and 4000 is, respectively, 30.25%, 9.5%, and 1.4%. The simulations are based on linear memristors (model 1 in Table 1).

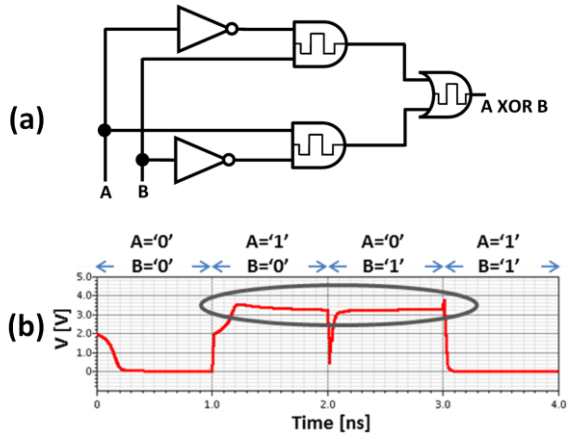


Figure 7. Two-input MRL XOR. (a) A schematic of MRL XOR, consisting of one memristor-based OR, two memristor-based AND, and two CMOS inverters, and (b) dynamic behavior of an MRL XOR logic gate. The high voltage is 4 volts. The output voltage degrades by approximately 15% for the input cases of ('1', '0') and ('0', '1').

CMOS gate capacitance which increases the delay. The actual delay is higher than the delay determined by (3) and shown in Figure 4b, although this difference is relatively small (0.1% to 4%). The degradation of each logic gate can be determined from (1) and (2).

MRL logic gates can be inserted into a standard cell library as in standard CMOS logic. These standard cell libraries can consist of NOR and NAND logic gates, where memristive-based OR (AND) and a CMOS inverter produces the functionality of a NOR (NAND) logic gate. Using NOR and NAND as standard cells overcomes the degradation issue since no current flows from the output node during steady state as the output node of the AND (OR) logic gate is connected to a CMOS gate. In this approach, each standard cell requires two connections between the CMOS and memristive layers. This approach is robust and relatively simple to design, albeit less efficient in terms of power consumption and area as compared to a circuit, where the CMOS inverter is only applied when signal restoration is needed or when the logic function requires signal inversion (an *optimized approach*).

IV. Eight-bit Full Adder Case Study

To investigate the MRL family, an eight-bit full adder is considered as a case study. Five different parameter sets of memristive devices are chosen to evaluate a variety of memristive characteristics. The primary parameters are the linearity coefficient and the current threshold (α_{on}, α_{off}, i_{on}, and i_{off} in the TEAM model [6]) varying from a linear memristor with no threshold, *i.e.*, the change in the resistance is linearly dependent on the current, to a nonlinear memristor with a current threshold. All of the other parameters are chosen to exhibit hysteretic behavior and cannot be therefore numerically compared to CMOS-only logic. The parameters for the memristive devices are listed in Table 1. In this case study, the standard cell approach and the optimized approach are designed with 0.12 µm CMOS and simulated in SPICE. Schematics of the one-bit full adders used in this case study for

TABLE 1. DIFFERENT PARAMETERS OF THE MEMRISTIVE DEVICES USED IN THE CASE STUDY

Device / Parameter	Linear with no current thresh-old	Linear with current thresh-old	Low non-linearity	Non-linear	Highly non-linear
Parameter set number	1	2	3	4	5
$\alpha_{off} = \alpha_{on}$	1	1	3	5	10
$i_{off} = -i_{on}$	100 fA	20 μA	5 μA	5 μA	10 μA
$k_{off} = -k_{on}$	$5 \cdot 10^{-8}$	10	0.1	0.01	0.001
R_{on}	1 kΩ				
R_{off}	100 kΩ				

both approaches are shown in Figure 8. The optimized approach saves CMOS transistors (eight instead of 18 per one-bit full adder) and vias (six instead of 18 per one-bit full adder).

The eight-bit full adder in this case study is achieved using eight cascaded one-bit full adders. A tradeoff between signal integrity and minimizing the number of vias is the primary design issue. To produce a distinct value for the output of the eight-bit full adder (S_i for i = 1, …, 8 and C_{OUT}), a set of CMOS buffers is added to the circuit to amplify the output signal. For the intermediate signals ($C_{OUT} \rightarrow C_{IN}$), no constraint is placed on the signal other than to maintain the correct logical polarity.

For parameter sets 1, 3 and 4 (memristive devices with a relatively low current threshold), the one-bit full adder shown in Figure 8 exhibits correct logical functionality, which requires amplifying the signal between different bit levels. Parameter sets 2 and 5 demonstrate a high current threshold and are therefore more sensitive to signal degradation due to partial switching. For these parameter sets, the circuit fails for all CMOS compatible voltages. Hence, for parameter sets 2 and 5, buffers have been added to each one-bit full adder to ensure correct logical behavior. The required voltage levels and number of components for each parameter set are listed in Table 2. The normalized power consumption[1] for each parameter set is listed in Table 3.

Note from the data listed in Tables 2 and 3 that unlike most digital applications [12], a linear memristive device with no threshold (as in parameter set 1) is preferable. Linear memristive devices minimize the number of connections between the CMOS and memristive layers and reduce power. Furthermore, the delay time of the MRL gates with linear memristive devices is relatively small, as shown in Figure 4a. Using nonlinear memristive devices requires a higher voltage, slower clock, and greater area due to the additional CMOS buffers and vias. The high voltage significantly increases the power consumption.

The optimized approach minimizes the delay and the number of vias, and consumes less dynamic power as compared to a standard cell library. This approach consumes, however, more total energy since the static power is non-zero.

[1] The power is normalized since the parameter set of the memristive devices is not correlated to a specific CMOS process.

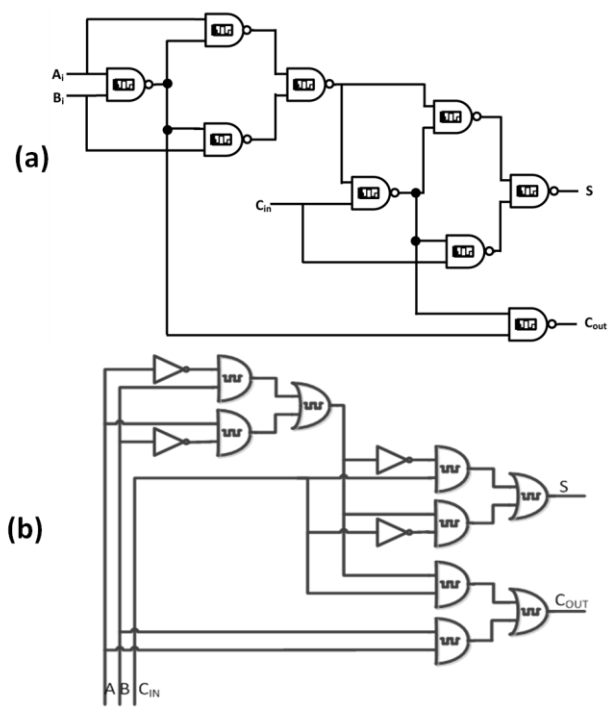


Figure 8. Schematic of an MRL one bit full adder used in the case study (S = XOR[A, B, C_{IN}], $C_{OUT} = A \cdot B + C_{IN} \cdot$XOR[$A$, B]); (a) for the standard cell approach and (b) the optimized approach. The standard cell approach requires 18 memristors, 18 CMOS transistors, and 18 vias. The optimized approach, however, requires 18 memristors and only eight CMOS transistors and six vias.

TABLE 2. SUMMARY OF CASE STUDY

Parameter set	Supply voltage [Volt]	Number of memristors	Number of CMOS transistors	Number of vias
CMOS – based	1	-	288	-
Standard cell	1	144	144	144
1	1	144	160	80
2	6.5	144	228	96
3	3	144	128	80
4	4	144	160	80
5	6.5	144	256	96

TABLE 3. POWER CONSUMPTION AND ENERGY FOR CASE STUDY

Parameter set	Average power [normalized]	Total energy [normalized]
Standard cell approach (for parameter set 1)	1	1
1	0.72	5.02
2	386.4	2035.1
3	22.5	167.9
4	60.8	499.2
5	354.95	2004.5

V. CONCLUSIONS

The advantages of combining memristors and CMOS transistors are shown in this paper. Memristor Ratioed Logic

(MRL), a hybrid CMOS-memristive logic family, is described. The compatibility of memristors and CMOS is exploited to increase logic density. A design example is also described saving approximately 50% in area as compared to CMOS logic.

A linear memristive device with no current threshold is shown to be preferable for the MRL logic family, unlike other digital applications, where a threshold and nonlinearity are desirable. MRL gates based on linear memristive devices are faster, smaller, and consume less power as compared to nonlinear memristive devices.

The Memristor Ratioed Logic family opens opportunities for additional hybrid memristive/CMOS integrated circuit structures to increase logic density. Although standalone CMOS logic is preferable in terms of performance as compared to MRL, a hybrid approach can further extend CMOS technology and enhance computational abilities for next generation digital integrated circuits.

REFERENCES

[1] S. E. Thompson and S. Parthasarathy, "Moore's Law: the Future of Si Microelectronics," *Materials Today*, Vol. 9, No. 6, June 2006.

[2] V. F. Pavlidis and E. G. Friedman, *Three-Dimensional Integrated Circuit Design*, Morgan Kaufmann, 2009.

[3] L. O. Chua, "Memristor – the Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[4] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209-223, February 1976.

[5] P. Vontobel, W. Robinett, J. Straznicky, P. J. Kuekes, and R. S. Williams, "Writing to and Reading from a Nano-Scale Crossbar Memory Based on Memristors," *Nanotechnology*, Vol. 20, No. 42, pp. 1-21, October 2009.

[6] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.

[7] J. Borghetti, Z. Li, J. Strasnicky, X. Li, D. A. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A Hybrid Nanomemristor/Transistor Logic Circuit Capable of Self-Programming," *Proceedings of the National Academy of Sciences*, Vol. 106, No. 6, pp. 1699-1703, February, 2009.

[8] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-Nanosecond Switching of a Tantalum Oxide Memristor," *Nanotechnology*, Vol. 22, No. 48, pp. 1-7, November 2011.

[9] J. J. Yang *et al*, "High Switching Endurance in TaOx Memristive Devices," *Applied Physics Letters*, Vol. 97, No. 23, pp. 1-3, December 2010.

[10] J. Nickel, "Memristor Materials Engineering: From Flash Replacement Towards a Universal Memory," *Proceedings of the IEEE IEDM Advanced Memory Technology Workshop*, December 2011.

[11] G. Snider, "Computing with Hysteretic Resistor Crossbars," *Applied Physics A*, Vol. 80, No. 6, pp. 1165-1172, March 2005.

[12] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY Logic Design Procedure," *Proceedings of the IEEE International Conference on Computer Design*, pp. 142-147, October 2011.

[13] Q. Xia *et al*., "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic," *Nano Letters*, Vol. 9, No. 10, pp. 3640-3645, August 2009.

[14] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "An Energy-Efficient Memristive Threshold Logic Circuit," *IEEE Transactions on Computers*, Vol. 61, No. 4, pp.474-487, April 2012.

[15] L. O. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A: Materials Science & Processing*, Vol. 102, No. 4, pp. 765-783, March 2011.

[16] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI)*, (in press).

Logic Operations in Memory Using a Memristive Akers Array

Yifat Levy[*], Jehoshua Bruck[**], Yuval Cassuto[*], Eby G. Friedman[***], Avinoam Kolodny[*], Eitan Yaakobi[**], and Shahar Kvatinsky[*]

* Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel, e-mail:{yifatl@tx, ycassuto@ee, kolodny@.ee, skva@tx}.technion.ac.il

** Electrical Engineering, California Institute of Technology, Pasadena, CA 91125, e-mail: {bruck, yaakobi}@caltech.edu

*** Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627 USA. e-mail: friedman@ece.rochester.edu

***Abstract* – In-memory computation is one of the most promising features of memristive memory arrays. In this paper, we propose an array architecture that supports in-memory computation based on a logic array first proposed in 1972 by Sheldon Akers. The Akers logic array satisfies this objective since this array can realize any Boolean function, including bit sorting. We present a hardware version of a modified Akers logic array, where the values stored within the array serve as primary inputs. The proposed logic array uses memristors, which are nonvolatile memory devices with noteworthy properties. An Akers logic array with memristors combines memory and logic operations, where the same array stores data and performs computation. This combination opens opportunities for novel non-von Neumann computer architectures, while reducing power and enhancing memory bandwidth.**

***Keywords:* memristor, memristive systems, logic array, memory array, von Neumann architecture, Akers logic array.**

I. INTRODUCTION

Conventional computers are based on a von Neumann architecture, where separate units process and store data. A different approach is to process data within the same unit that stores the data (*i.e.*, process data within memory). An illustration of both architectures is shown in Figure 1. In this paper, a hardware version of processing

within memory is proposed. The proposed circuit is based on a study of rectangular logic arrays, first proposed in 1972 by Sheldon Akers [1].

In an *Akers logic array* (or, in short, an *Akers array*), the execution of any Boolean function is performed by flowing data across an array of primitive logic cells. The data are transferred from each primitive logic cell to neighboring cells, as shown in Figure 2a. The operation of an Akers array is similar to systolic array [2] and cellular automata [19]. The primitive logic cell has three inputs and two outputs, as shown in Figure 2b. The inputs of the primitive logic cell include two control inputs x and y and a variable input z, which is replaced in our circuit by an internal state (*i.e.*, the stored data). The primitive logic cell performs a predefined logical operation $f(x, y, z)$, which is described below. The output of each primitive logic cell is used as control inputs x and y of, respectively, the bottom and right neighboring primitive logic cells.

To execute any Boolean function within an Akers array, specific input values are inserted as control inputs into the left-most column and the upper-most row. The control input y of the left-most column is set to 1 for all rows, and the control input x of the upper-most row is set to 0 for all columns, as shown in Figure 2a. These control inputs along with the array structure and the function $f(x, y, z)$ determine the Boolean function computed by the array. The inputs to this Boolean function are the bits stored within the array cells. The output of the Boolean function computed by the Akers array is the output of the primitive logic cell at the bottom right of the array. It is also possible to define multiple Boolean functions (or, alternatively, a multi-bit output) on the same Akers array, in which case additional primitive cell outputs are used as external functional outputs. To date, an Akers array has been treated as a mathematical concept since the benefit of an Akers logic array with conventional semiconductor technology (*i.e.*, CMOS technology) is limited, as described in Section II.

The emergence of memristive technologies [3] enables the integration of computation and memory, including logic within memory [5-6, 20-26]. The high density of memristors and compatibility with CMOS makes an Akers array with memristors practical. In this paper, a memristive Akers array is proposed, where the variables z are stored within the memristive cells, and the control inputs x and y are

voltages. The proposed memristive Akers array serves as a practical example of in-memory computation.

The design of the proposed memristive Akers array is demonstrated here by a small example of a four by four array, producing a variety of array operations, including a bit sorting algorithm as a case study. The rest of the paper is organized as follows. In Section II, background describing both the Akers array and memristors is provided. The proposed memristive Akers array is described and evaluated in, respectively, Sections III and IV, followed by a discussion of design considerations for larger arrays in Section V. A small example of different array operations is described in Section VI, followed by some concluding remarks in Section VII.

II. BACKGROUND

In this section, the theory of the original Akers logic arrays is described and the basic principles of memristive devices are reviewed, including the model used in this paper for evaluating the proposed memristive Akers array.

A. Akers Logic Array

An Akers logic array is a two-dimensional array of identical primitive logic cells connected in a rectangular grid, as shown in Figure 2a. The primitive logic cell in the array is a three input logic gate that executes the logical operation,

$$f(x, y, z) = x\bar{z} + yz. \quad (1)$$

Note that in the original Akers array [1], four alternative logical operations that generate the correct behavior of the array are proposed. In this paper, only (1) is used due to the easy implementation with memristors.

The output of each primitive logic cell is transferred to the two neighboring primitive logic cells in the array – one below and one to the right of the array. The transferred data are the *x* and *y* control inputs of, respectively, the vertical and horizontal neighbors, as shown in Figure 2a. The control input *y* of the left-most column is set to 1 for all rows, and the control input *x* of the upper-most row is set to 0 for all columns.

The execution of a Boolean function is performed by organizing the contents of the array cells according to the particular specification, and reading the functional output from the output of the lower-right cell (or from multiple cell outputs in the case of a

Boolean function with a multiple bit output or, alternatively, multiple Boolean functions simultaneously computed within the same array). Hence, the same array can be used for different Boolean functions, each specifying a different organization of inputs. Examples of several Boolean functions are illustrated in Figure 3.

Sorting of four bits $\{z_0, z_1, z_2, z_3\}$ is shown in Figure 3a. The binary sorting function on n inputs is defined as the n Boolean functions $f_0, \ldots, f_{n-1}$, where $f_i(z_0, \ldots, z_{n-1}) = 1$ if the number of "1" inputs among $z_0, \ldots, z_{n-1}$ is greater than i (*i.e.*, f_0 is the maximum value and f_{n-1} is the minimum of the output). For the sorting function, each input variable of the sorting Boolean function is replicated a number of times up to the number of inputs [1]. For example, z_3 is replicated four times, while z_1 is replicated two times. The number of primitive logic cells is therefore $\sum_{i=0}^{n-1}(i + 1) = \frac{n^2}{2} + \frac{n}{2}$ where n is the number of inputs to the sorting Boolean function. The output bits of the sorting Boolean function are placed along the diagonal of the array, as shown in Figure 3a.

Another example for a Boolean function within an Akers array is a four-bit XOR [1], as shown in Figure 3b. The variable inputs of the primitive logic cells are arranged similarly to the sorting array, where the complementary value of the XOR inputs are also stored as input variables of the primitive logic cells. The output of the XOR operation is the output of the bottom right primitive logic cell. The number of primitive logic cells for an n-bit XOR is n^2.

Since the inputs of the Boolean function must be replicated within an array, the number of primitive logic cells increases quadratically with the number of inputs of the Boolean function. A CMOS Akers logic array therefore requires significant area, making an Akers array impractical with standard CMOS. In contrast, the density and circuit architecture of memristive devices make the Akers array natural for memories. A memristive Akers array within memory can be denser than standard SRAM (without computation capabilities), as listed in Table 1.

B. Memristors

Memristors and memristive devices [3, 7] are two-port passive elements with varying resistance. The change in the resistance of these devices depends on the current flowing through the device (or, alternatively, the voltage across the device), as

shown in Figure 4. While in theory the change in the resistance of a memristor depends directly on the current (or voltage), for memristive devices the dependence can be more complicated and described by internal state variables [7]. In this paper, the term *memristor* is used to describe both memristors and memristive devices.

Since 2008, numerous emerging nonvolatile memory technologies have been connected to the theory of memristors [8-12]. These technologies are nonvolatile, fast, dense, CMOS compatible, low power, and have high write endurance. The compatibility of memristors with CMOS enables the use of memristors not only as memory, but also as logic circuits [4-6, 13, 20-26].

Several models have been proposed to describe the behavior of memristors. In this paper, the TEAM model is used [14]. The TEAM model is general and can fit memristors from different technologies. In the TEAM model, it is assumed that a memristor has current thresholds, i_{off} and i_{on}, and an internal state variable x. When the current flowing through the memristor is above the current thresholds, the memristor changes state either from R_{on} to R_{off} or from R_{off} to R_{on} depending upon the original state and direction of the current. The voltage-current relationship and the change in state variable are described by

$$v(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{x_{off} - x_{on}}(x - x_{on})\right] \cdot i(t) \tag{2}$$

$$\frac{dx(t)}{dt} = \begin{cases} k_{off} \cdot \left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} \cdot f_{off}(x), & 0 < i_{off} < i, \quad (3a) \\ k_{on} \cdot \left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} \cdot f_{on}(x), & i < i_{on} < 0, \quad (3b) \\ 0, & otherwise, \quad (3c) \end{cases}$$

where R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, x_{on} and x_{off} are, respectively the minimum and maximum value of the state variable x, $f_{on}(x)$ and $f_{off}(x)$ are window functions (the TEAM window function is used in this paper), and k_{off}, k_{on}, α_{off}, and α_{on} are fitting parameters. An example of an I-V curve of the TEAM model is shown in Figure 5.

III. PROPOSED MEMRISTIVE AKERS LOGIC ARRAY

As previously mentioned, an Akers array with conventional CMOS technology is impractical due to the significant area requirements. The use of memristors, which are

dense and fabricated physically above the CMOS transistors, significantly reduces the area.

The proposed memristive Akers primitive logic cell is based on the structure of complementary memristors (or complementary resistive switches, CRS) [15, 16]. In the proposed memristive realization of an Akers array, the input variable *z* is the stored internal state of a memristor. The inputs of the executed Boolean function are therefore treated as stored data within a memristive memory array. In this section, the structure of the primitive logic cell is described as well as the operation of the array.

A. Primitive logic cell structure

The proposed primitive logic cell realizes the logical connectivity described by (1). The primitive cell consists of two anti-serial memristors (connected with opposite polarity), as shown in Figure 6a. The control inputs of the primitive logic cell *x* and *y* are voltages (logical one and zero are, respectively, a positive voltage V_r and ground). The variable input *z* is the stored logical state of memristor M_Z, which is represented by the resistance of the device (low and high resistances are considered, respectively, as logical one and zero). The memristor $M_{\bar{Z}}$ has the complementary logical state of M_Z. The stored logical state of M_Z and $M_{\bar{Z}}$ are written during a write operation prior to execution.

Ideally, the memristors can be modeled as switches, where a high resistance is an open circuit and a low resistance is a short circuit, as shown in Figure 6b. In an ideal model, one switch is open and the other switch is closed. If *z* is logical one, the switch of *z* is closed and the logical value of *y* is transferred to the output. If *z* is logical zero, the switch is open and the complementary switch is closed, transferring *x* to the output.

The precise output of the primitive logic cell is the result of a voltage divider between M_Z and $M_{\bar{Z}}$. The output voltage V_f is

$$V_f = \frac{V_y - V_x}{R_Z + R_{\bar{Z}}} \cdot R_{\bar{Z}} + V_x, \tag{4}$$

where R_Z and $R_{\bar{Z}}$ are, respectively, the resistance of memristors M_Z and $M_{\bar{Z}}$, varying from R_{on} to R_{off}. V_x and V_y are the input voltages *x* and *y*. The output voltage V_f for

different input conditions is listed in Table 2, demonstrating that, as required, the primitive logic cell indeed executes the Boolean function (1).

B. Logic array operation

The Akers logic array is an array of primitive logic cells that can also be used as a memory array, as shown in Figure 7. Unlike regular memory arrays, the memristive Akers logic array can compute different Boolean functions in addition to storing data. The computation of Boolean functions within the logic array is divided into two stages. The initial stage is a "write" operation to the memristors. In this stage, the initial logical state of memristors M_Z and $M_{\bar{Z}}$ is simultaneously written. This stage can be part of a regular write operation of the memory or, alternatively, an explicit initialization prior to computing the Boolean function. In this paper, initialization of a single primitive logic cell is evaluated. Writing to the array (*e.g.*, addressing the specific primitive cells within the array and parallelizing the writes) is only briefly discussed since this process is similar in any CRS-based memory (*e.g.*, see [16]). Relevant adjustments (*e.g.*, adding CMOS selectors to achieve isolation between the primitive cells and maintain regular read and write operations), however, need to be performed to achieve a memory integrated with an Akers logic array, as shown in Figure 7c.

The second stage executes the Boolean function. In this stage, a low voltage is used to ensure that the resistance of the memristors in the array does not change.

1) Stage 1 – initialization of the primitive logic cells (write)

Initialization of the logical states of M_Z and $M_{\bar{Z}}$ is simultaneously achieved due to the anti-serial connection of both memristors. In the complementary structure, applying a sufficiently high voltage to both memristors switches both memristors to different resistances, where one memristor achieves a high resistance and the other memristor achieves a low resistance. The write procedure in a complementary pair of memristors is shown in Figure 8.

To write a logical one to M_Z, the resistances M_Z and $M_{\bar{Z}}$ are required to be, respectively, a low and high resistance. The write procedure therefore applies a sufficiently positive voltage V_w to y while grounding x. To write a logical zero to M_Z, the write procedure applies V_w to x while grounding y, or alternatively, apply $-V_w$ to y and grounding x. At the end of the write operation, the resistance of $M_Z$and $M_{\bar{Z}}$ are

R_{ON} and R_{OFF}, where the resistance of one memristor is R_{ON} and the resistance of the other memristor is R_{OFF}.

2) Stage 2 – execution of the Boolean function (read)

The structure of the memristive logic array is shown in Figure 2a. The array is similar to the structure of the original Akers logic array. In a memristive Akers logic array, each primitive logic cell consists of complementary memristors. The *x* and *y* control inputs are voltages, and, as in the original Akers array, the input *y* of the left-most column is set to logical one (execution voltage V_r), and the input *x* of the upper row is set to logical zero (ground) for all columns. Since the output of the memristive primitive logic cell is a voltage, the result of the logical operation for each primitive logic cell is transferred to the neighboring cells.

To maintain correct operation of the memristive Akers logic array, the resistance of the memristors in the array must not change during execution. The current flowing through the memristors I_r is therefore maintained lower than the threshold current of the memristors. The current is

$$I_r = \frac{|V_y - V_x|}{R_Z + R_{\bar{Z}}} \leq \frac{V_r}{R_{ON} + R_{OFF}} < \max\left(|i_{off}|, |i_{on}|\right). \quad (5)$$

IV. EVALUATION OF PRIMITIVE LOGIC CELLS

In this section, the proposed memristive primitive logic cell is evaluated with 0.18 µm CMOS and simulated in SPICE. A Verilog-A TEAM model [17] is used to simulate the behavior of the memristors.

The primitive logic cell is based on a complementary resistive switch structure. The CRS behaves as a linear resistor with a resistance of R_{ON} + R_{OFF} below a certain voltage. Above this voltage, hysteresis exists in the current-voltage curve of the CRS [15, 16]. The current-voltage curve of the primitive logic cell is shown in Figure 9.

The primitive logic cell is evaluated with and without CMOS selectors connected to the control inputs, *x* and *y*. The primitive logic cell drives a load capacitor of 10 fF. The parameters used for the memristors are listed in Table 3. A schematic of the simulated primitive logic cell is shown in Figure 10a. The results of the initializing stage are shown in Figure 10b. The write latency of the primitive cell depends upon the switching time of the memristor, assumed as 1.1 ns. The primitive logic cell

exhibits a write latency of 6.6 ns (six times more than the switching time of a single memristor).

The results of the execution stage are shown in Figures 10c and 10d. The primitive logic cell executes the correct logical behavior with degradation in the output signal. The degradation depends upon the ratio between R_{OFF} and R_{ON}. The output degradation is 0.1% without selectors (R_{OFF}/R_{ON} = 1000) and 4% with CMOS selectors (for a 0.18 µm CMOS process). The output degradation is discussed in the following section.

V. OUTPUT DEGRADATION

Since memristors are passive elements, signal degradation occurs at the output of each primitive logic cell. The degradation depends primarily on the ratio between R_{OFF} and R_{ON}, where a higher ratio reduces the degradation. The degradation limits the size of the Akers array.

The degradation of the output signal as a function of array size is shown in Figure 11a for Akers arrays with and without CMOS selectors. The use of CMOS selectors makes the output degradation worse since the CMOS element adds a resistance in series. For larger arrays, the degradation is more significant and limits the size of the sub-arrays of the memory. The degradation for different ratios of R_{OFF} and R_{ON} is shown in Figure 11b. For an array composed of 128 by 128 primitive logic cells, the minimal degradation of the output reaches 10% for R_{OFF}/R_{ON} = 1000. For arrays with CMOS selector with a resistance of 1 kΩ, the actual output degradation is 15%. Using larger CMOS transistors lowers the degradation. A higher R_{OFF}/R_{ON} ratio enables a larger array, where a ratio of 10,000 enables arrays of more than a million logic primitive cells with an output degradation of 10%.

VI. TEST CASE – MEMRISTOR-BASED LOGIC WITHIN MEMORY ARRAY

To evaluate a memristive Akers array, several Boolean functions are investigated within the array. In this section, simulation results of a two-input XOR and sorting of four bits are presented as simple test examples.

A. Two-input XOR

The schematic and array structure of a $XOR(A,B)$ are shown in Figure 12. The memristive Akers array is a two by two array, consisting of eight memristors. Initializing the array (writing the inputs to the memristors) is achieved prior to execution. The execution is evaluated with the same parameters listed in Table 3, exhibiting the correct output. The average and maximum output degradation are, respectively, 20% and 31% for a two-input XOR with 0.18 μm CMOS selectors (3% without selectors). The relatively high degradation is due to the minimal size of the CMOS selectors and the use of high voltage transistors, which have a relatively high resistance. As previously mentioned, increasing the width of the transistors significantly lowers the signal degradation.

The average power of the array during execution is, respectively, 6.2 μW and 33.6 μW without and with CMOS selectors. The results for different input conditions are shown in Figure 13. For small arrays, adding CMOS selectors does not affect the speed of the circuit. For an array with CMOS selectors, execution is slower due to the capacitance of the selectors.

B. Sorting of bits

To evaluate sorting of bits, a four-bit sorting Boolean function is executed within the memristive Akers array. The memristive Akers array consists of ten primitive logic cells (see Figure 3a) and 20 memristors. The execution is evaluated with the same parameters listed in Table 3, showing correct output and an average output degradation of 0.3% without CMOS selectors. The average power of the array during execution is 1.6 μW. Results for different input conditions are shown in Figure 14.

VII. CONCLUSIONS

The proposed memristive Akers array contains a pair of complementary memristors in each cell. The array can therefore be used as a memristive memory, where a single bit is stored within a memristor pair rather than a single memristor [15, 16]. Each cell also performs a primitive Boolean operation, which enables the logic functionality of the array, as initially shown by Akers. The combination of an Akers array and memory is promising and may lead to additional uses, as described in [18]. For example, an Akers logic array naturally performs bit sorting which may lead to efficient sorting of words and other data structures.

The integration of memristive memory with a logic array that executes any Boolean function can lead to a variety of novel non-von Neumann architectures. The Akers array architecture eliminates the memory bottleneck, reducing power and bandwidth. Memristive Akers logic arrays may also be beneficial for image processing applications and error correcting operations within memory.

Acknowledgements

The authors thank Ravi Patel of the University of Rochester for his useful comments.

REFERENCES

[1] S. B. Akers, Jr., "A Rectangular Logic Array," *IEEE Transactions on Computers*, Vol. C-21, No. 8, pp. 848-857, August 1972.

[2] H. T. Kung, "Why Systolic Architectures?" *IEEE Computers*, Vol. 15, No. 1, pp. 37-46, January 1982.

[3] L. O. Chua, "Memristor – The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[4] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic," *Proceedings of the International Cellular Nanoscale Networks and their Applications*, pp. 1-6, August 2012.

[5] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI)* (in press).

[6] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-Based IMPLY Logic Design Procedure," *Proceedings of the IEEE International Conference on Computer Design*, pp.142-147, October 2011.

[7] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209- 223, February 1976.

[8] D. B. Strukov, G. S.Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, Vol. 453, pp. 80-83, May 2008.

[9] X. Wang, Y. Chen, H. Xi, and D. Dimitrov, "Spintronic Memristor through Spin-Torque-Induced Magnetization Motion," *IEEE Electron Device Letters*, Vol. 30, No. 3, pp. 294-297, March 2009.

[10] L. O. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A: Materials Science & Processing*, Vol. 102, No. 4, pp. 765-783, March 2011.

[11] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories – Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, Vol. 21, Issue 25-26, pp. 2632-2663, July 2009.

[12] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive Devices for Computing," *Nature Nanotechnology*, Vol. 8, pp. 13-24, January 2013.

[13] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic for Hybrid CMOS-Memristor Circuits," *IEEE Transactions on Nanotechnology* (in review).

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.

[15] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary Resistive Switches for Passive Nanocrossbar Memories," *Nature Materials*, Vol. 9, No. 5, pp. 403–406, April 2010.

[16] O. Kavehei, S. Al-Sarawi, S., K.-R. Cho, K. Eshraghian, and D. Abbott, "An Analytical Approach for Memristive Nanoarchitectures," *IEEE Transactions on Nanotechnology*, Vol. 11, No. 2, pp. 374-385, March 2012.

[17] S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Models of Memristors for SPICE Simulations," *Proceedings of the IEEE Convention of Electrical and Electronics Engineers in Israel*, pp. 1-5, November 2012.

[18] E. Yaakobi, A. Jiang, and J. Bruck, "In-Memory Computing of Akers Logic Array," *Proceedings of the IEEE International Symposium on Information Theory*, pp. 2369-2373, July 2013.

[19] S. Wolfram, "Universality and Complexity in Cellular Automata," *Physica D: Nonlinear Phenomena*, Vol. 10, No. 1-2, pp. 1-35, January 1984.

[20] E. Gale, B. de Lacy Costello, and A. Adamatzky, "Boolean Logic Gates from a Single Memristor via Low-Level Sequential Logic," *Proceedings of the International Conference on Unconventional Computation and Natural Computation*, pp. 78-89, July 2013.

[21] E. Gale, B. de Lacy Costello, and A. Adamatzky, "Is Spiking Logic the Route to Memristor-Based Computers?" *Proceedings of the International Conference on Electronics, Circuits and Systems*, pp. 297-300, December 2013.

[22] M. D. Pickett and R S. Williams, "Phase Transitions Enable Computational Universality in Neuristor-Based Cellular Automata," *Nanotechnology*, Vol. 24, No. 38, pp. 1-7, September 2013.

[23] S. Shin, K. Kim, and S.-M. Kang, "Memristive XOR for resistive multiplier," *Electronics Letters*, Vol. 48, No. 2, pp. 78-80, January 2012.

[24] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann - Logic Operations in Passive Crossbar Arrays Alongside Memory Operations," *Nanotechnology*, Vol. 23, No. 305205, August 2012.

[25] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable 'Stateful' Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873-876, April 2010.

[26] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC – Memristor Aided LoGIC," *IEEE Transactions on Circuits and Systems II: Express Briefs* (in review).

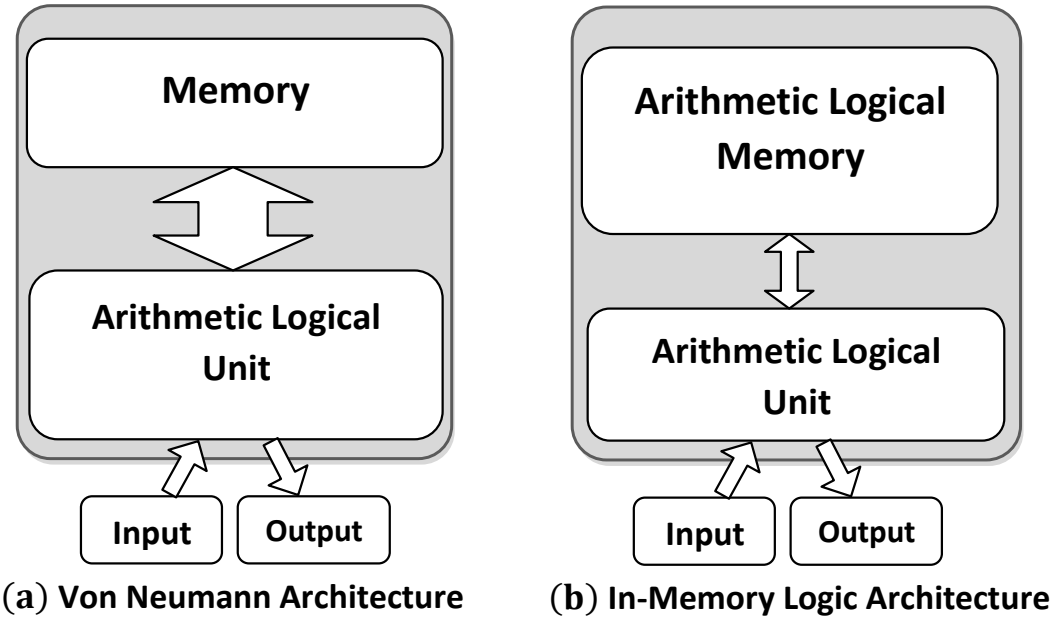


Figure 1. Different computer architectures. (a) von Neumann architecture – separate memory and an ALU. (b) Processing within memory architecture (*e.g.*, memristive Akers array). The memory can also process data. The size of the ALU is therefore smaller and the required memory bandwidth lower (schematically represented by the thickness of the arrow between the memory and the ALU).

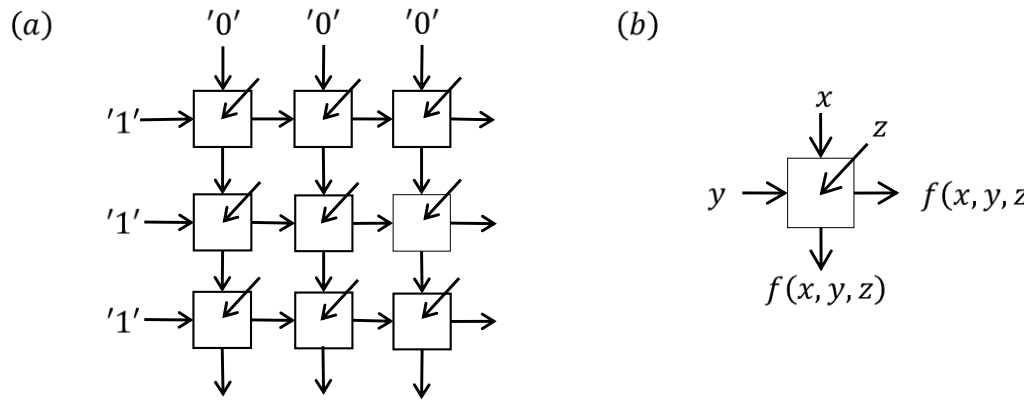
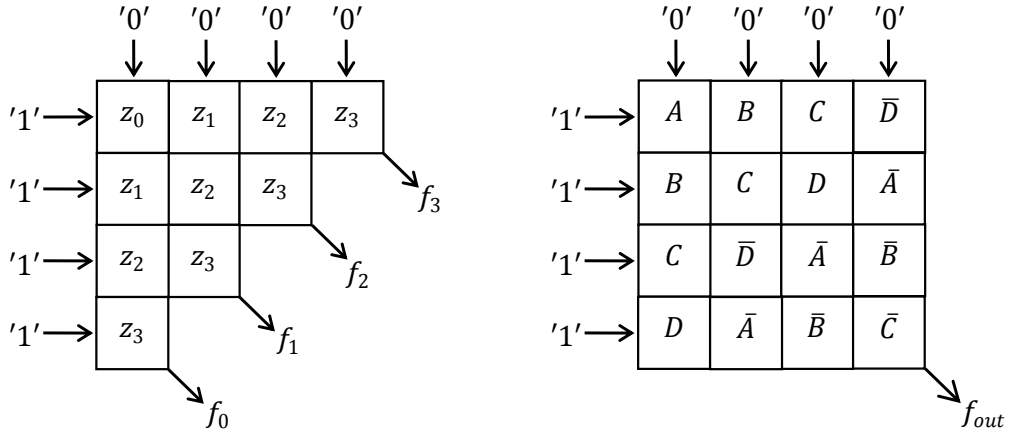


Figure 2. Akers logic array. (a) An example of a three by three Akers array structure. (b) A primitive logic cell with three inputs *x, y, z* and two identical outputs *f*(*x, y, z*).



(a) Array for sorting **(b) Array for $XOR(A, B, C, D)$**

Figure 3. Four-bit input structure for an Akers arrays for Boolean functions (a) Sort $\{z_0, z_1, z_2, z_3, \}$, and (b) $XOR(A, B, C, D)$.

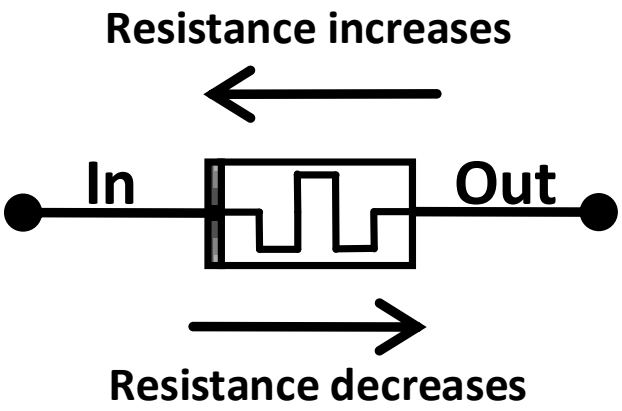


Figure 4. Memristor symbol. The polarity of the memristor is represented by the thick black line. When current flows into the device, the resistance of the device increases. When current flows out of the device, the resistance of the device decreases.

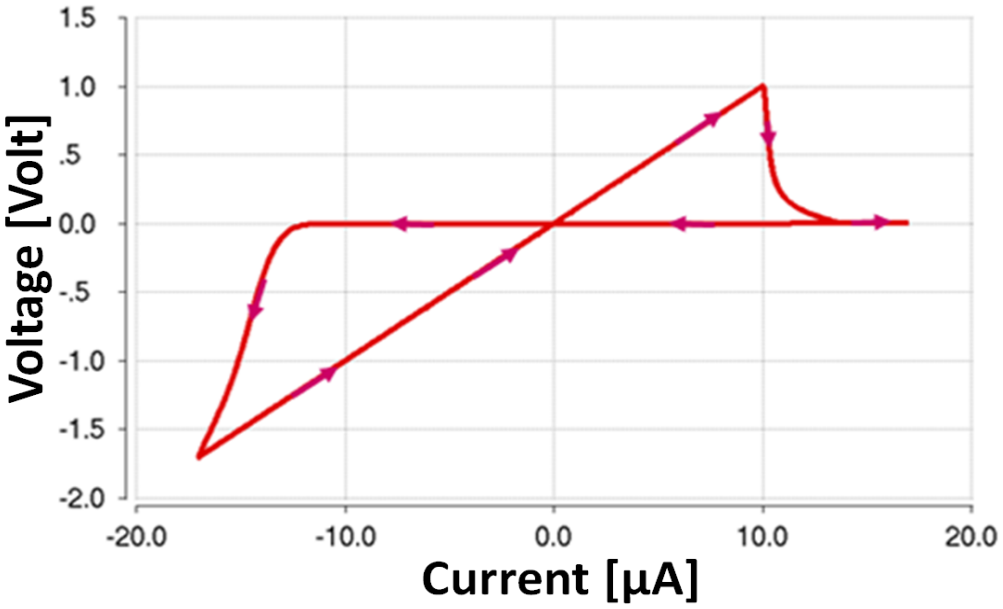


Figure 5. Current-voltage characteristics of a memristor based on the TEAM model [14] for a sinusoidal current input with an amplitude of 17 µA and frequency of 100 kHz. The memristor parameters are listed in Table 3.

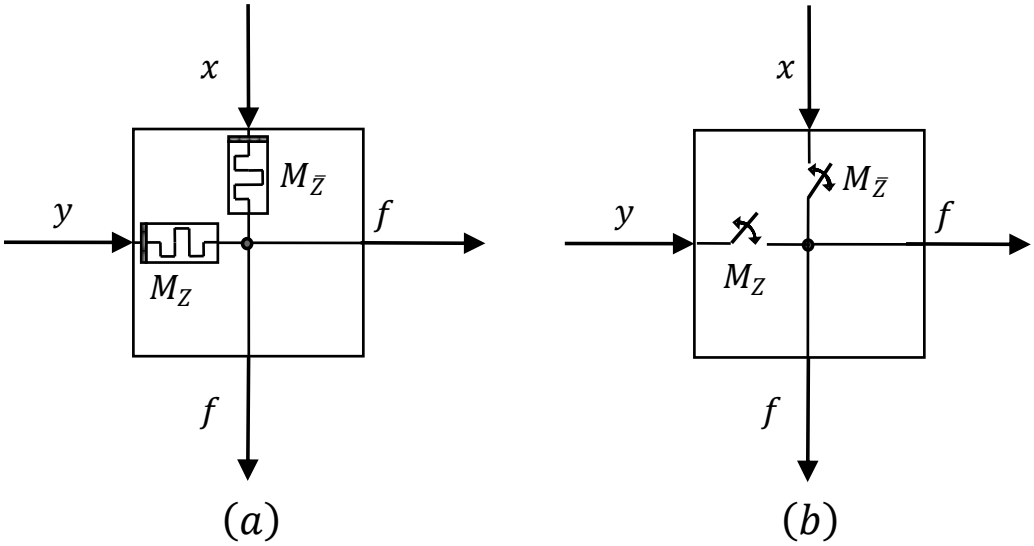


Figure 6. Primitive logic cell. (a) The proposed primitive logic cell using memristors. (b) A behavioral model of the basic logic cell, where the memristors are modeled as ideal switches.

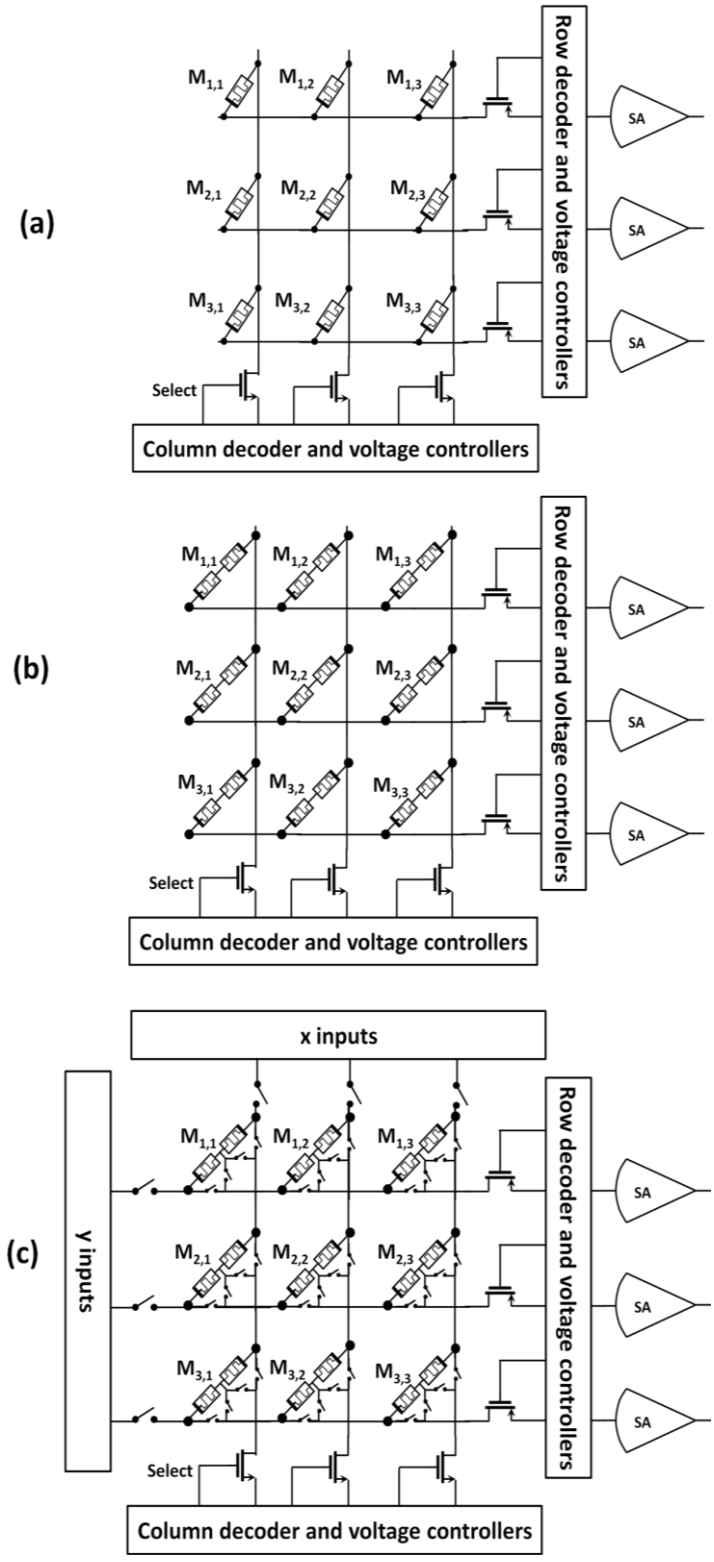


Figure 7. Memristive memories (exemplified by a three by three array). (a) Single memristor within a crossbar, (b) standard complementary memristive cells within a crossbar, and (c) Akers logic array within a memristive memory. The basic memory cell for the Akers logic array consists of two memristors and four CMOS transistors (as selectors).

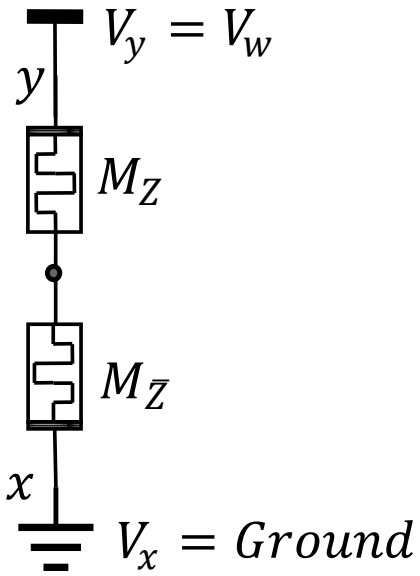


Figure 8. Write operation of logical one to memristor M_Z. Due to the complementary structure of the circuit, writing to M_Z and $M_{\bar{Z}}$ is achieved simultaneously in both memristors by applying a single voltage V_W. After the write procedure, the resistance of M_Z and $M_{\bar{Z}}$ is, respectively, R_{ON} and R_{OFF}.

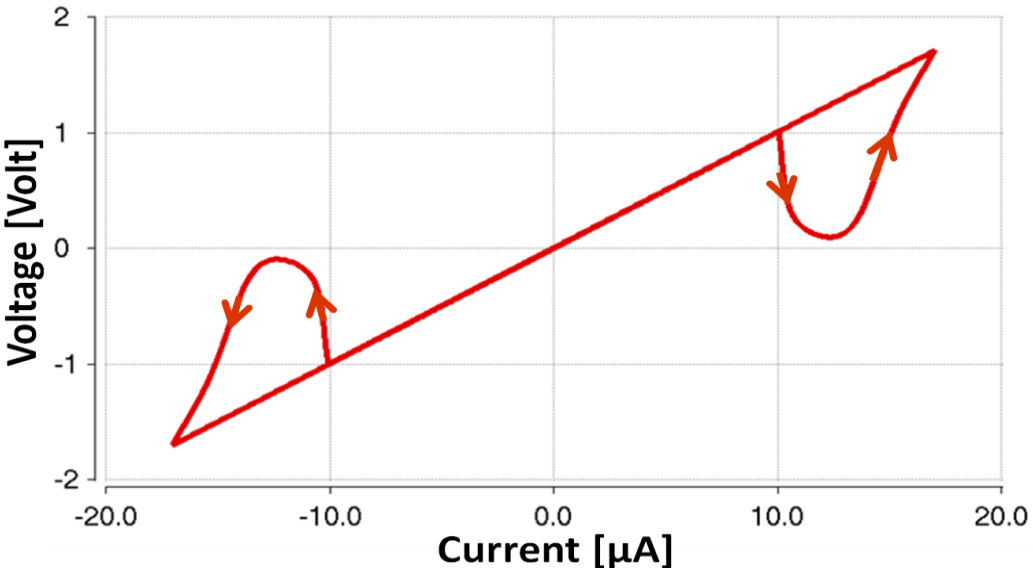


Figure 9. Current-voltage characteristic of the primitive logic cell for a sinusoidal current input with an amplitude of 17 µA and frequency of 100 kHz. The circuit parameters are listed in Table 2. For a current lower than the current thresholds i_{on} and i_{off} (10 µA), the resistance of both memristors is constant. For a current higher than the current thresholds, the resistance of both memristors changes.

TABLE 1. AREA OF MEMORY TECHNOLOGIES

F – feature size, T – transistor, C – capacitor, and R – resistive device (memristor)

Technology	Memory Cell	Area per Cell [F^2]	Computing Capabilities	Sequential
SRAM	6 T	140	No	---
DRAM	1T 1C	6-12	No	---
Flash	1 T	4	No	---
RRAM (memristive memory, single device per cell)	1 R	4	Yes [5-6, 25-26]	Yes
Complementary Resistive Switches	2 R	4-8	Yes [24]	Yes
Akers Array within Memory	2 R 4 T	20-90	Yes	No

TABLE 2. OUTPUT VOLTAGE OF PRIMITIVE LOGIC CELL FROM (4)

x	y	z	R_Z	V_f – derived from (4)	$f(x, y, z)$
0	0	0	R_{OFF}	$\mathbf{0}$	0
0	0	1	R_{ON}	$\mathbf{0}$	0
0	1	0	R_{OFF}	$\frac{R_{ON}}{R_{OFF}+R_{ON}} \cdot V_r$	0
0	1	1	R_{ON}	$\frac{R_{OFF}}{R_{OFF}+R_{ON}} \cdot V_r$	1
1	0	0	R_{OFF}	$\frac{R_{OFF}}{R_{OFF}+R_{ON}} \cdot V_r$	1
1	0	1	R_{ON}	$\frac{R_{ON}}{R_{OFF}+R_{ON}} \cdot V_r$	0
1	1	0	R_{OFF}	V_r	1
1	1	1	R_{ON}	V_r	1

TABLE 3. MEMRISTOR PARAMETERS

k_{on}	-8 m/sec
k_{off}	0.5 m/sec
i_{on}	-10 uA
i_{off}	10 uA
x_{on}	0
x_{off}	3 nm
α_{on}	1
α_{off}	4
R_{ON}	100 Ω
R_{OFF}	100 kΩ
V_w	3 V
V_r	1 V
CMOS Selectors	CMOS 0.18 µm process W = 0.42 µm

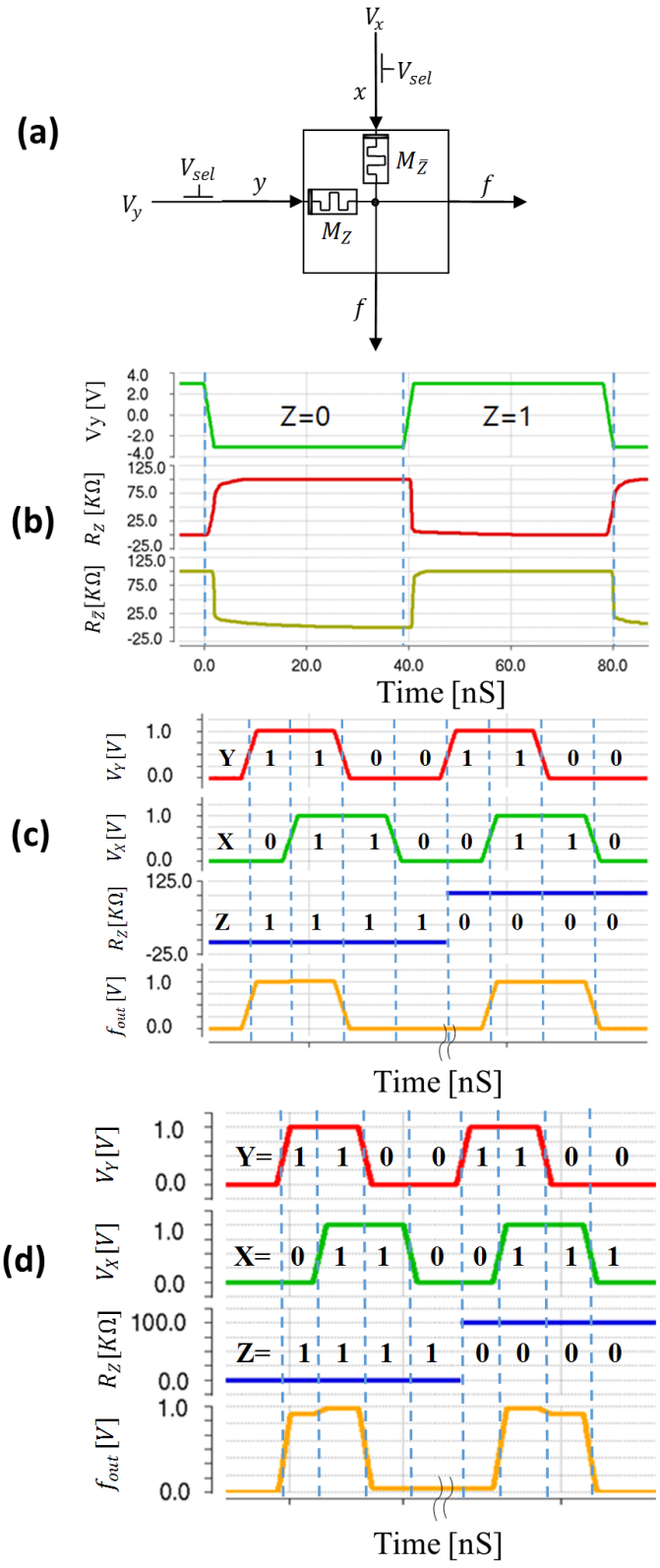


Figure 10. Initialization and execution of primitive logic cell. (a) Schematic of the simulated circuit, (b) simulation of memristive initialization operation. V_y is the write voltage applied to the primitive logic cell (positive and negative for, respectively, writing logical one and zero to Z), and simulation of memristor execution operation (c) without selectors and (d) with selectors. The simulation parameters are listed in Table 2.

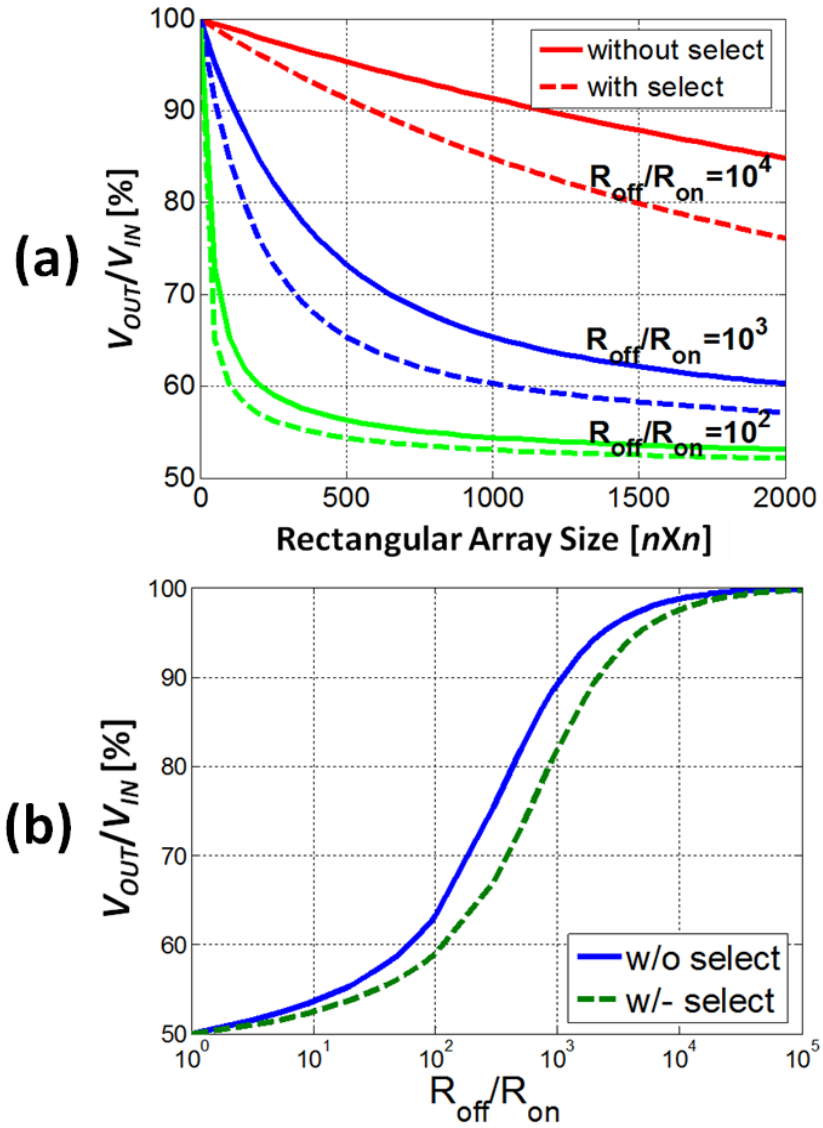


Figure 11. Output signal degradation for an Akers array with (dashed line) and without (solid line) CMOS selectors. (a) Signal degradation as a function of rectangular array size for different R_{OFF}/R_{ON} ratios (10^4 in red, 10^3 in blue, and 10^2 in green), and (b) signal degradation in rectangular array of 128 by 128 as a function of the resistance ratio R_{OFF}/R_{ON} with CMOS selector. R_{ON} = 1 kΩ, the resistance of a CMOS selector is 1 kΩ.

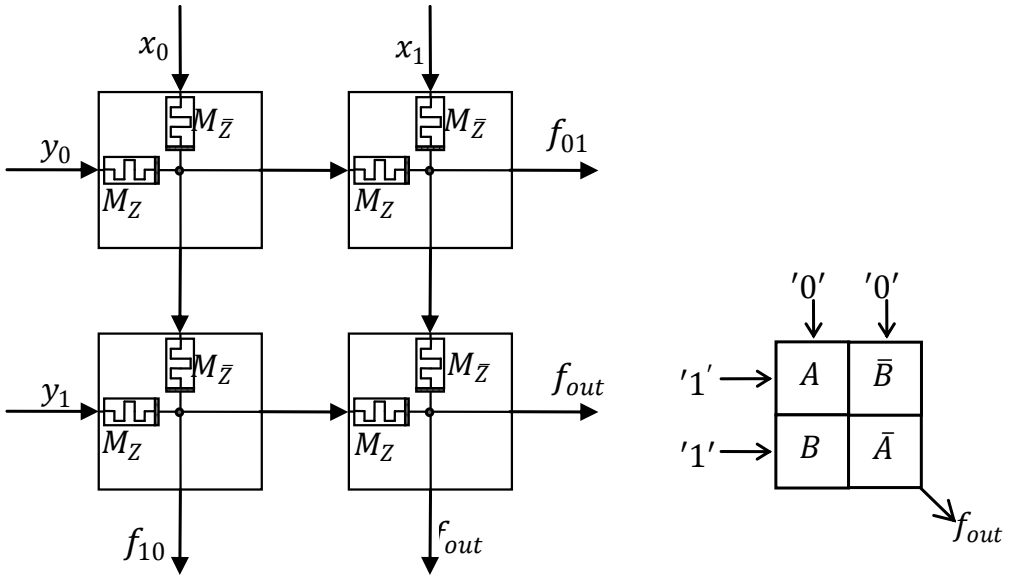


Figure 12. Two-input XOR. (a) Schematic of a two by two memristive Akers array, and (b) the array structure of the Boolean function XOR(*A*, *B*).

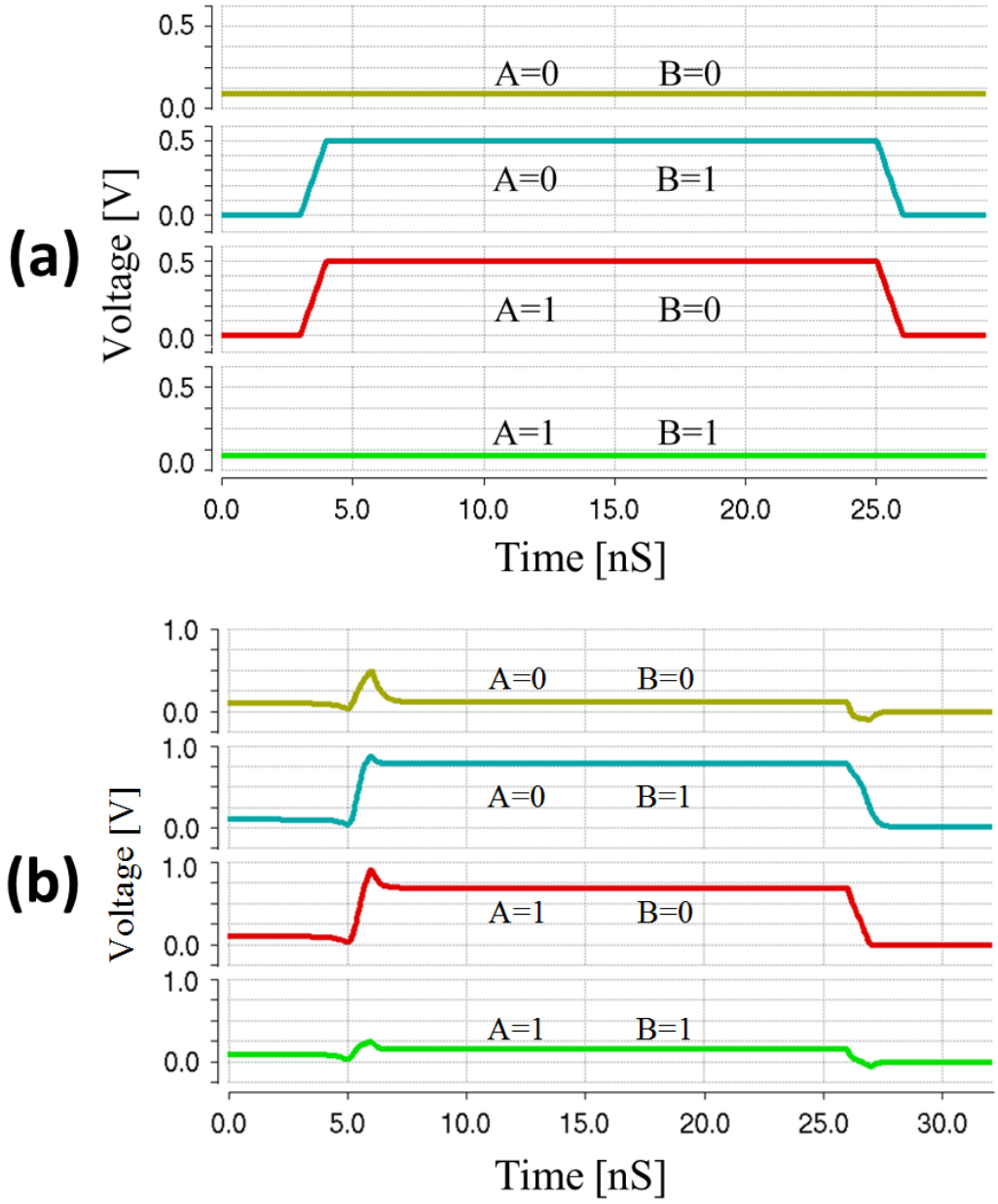


Figure 13. Simulation results of a two-input XOR (a) without CMOS selectors and (b) with CMOS selectors for different inputs *A* and *B*. The average output degradation is 3% and 20%, respectively, without and with CMOS selectors for a 0.18 µm CMOS process. The execution voltage V_r for the XOR without selectors is 0.5 volts.

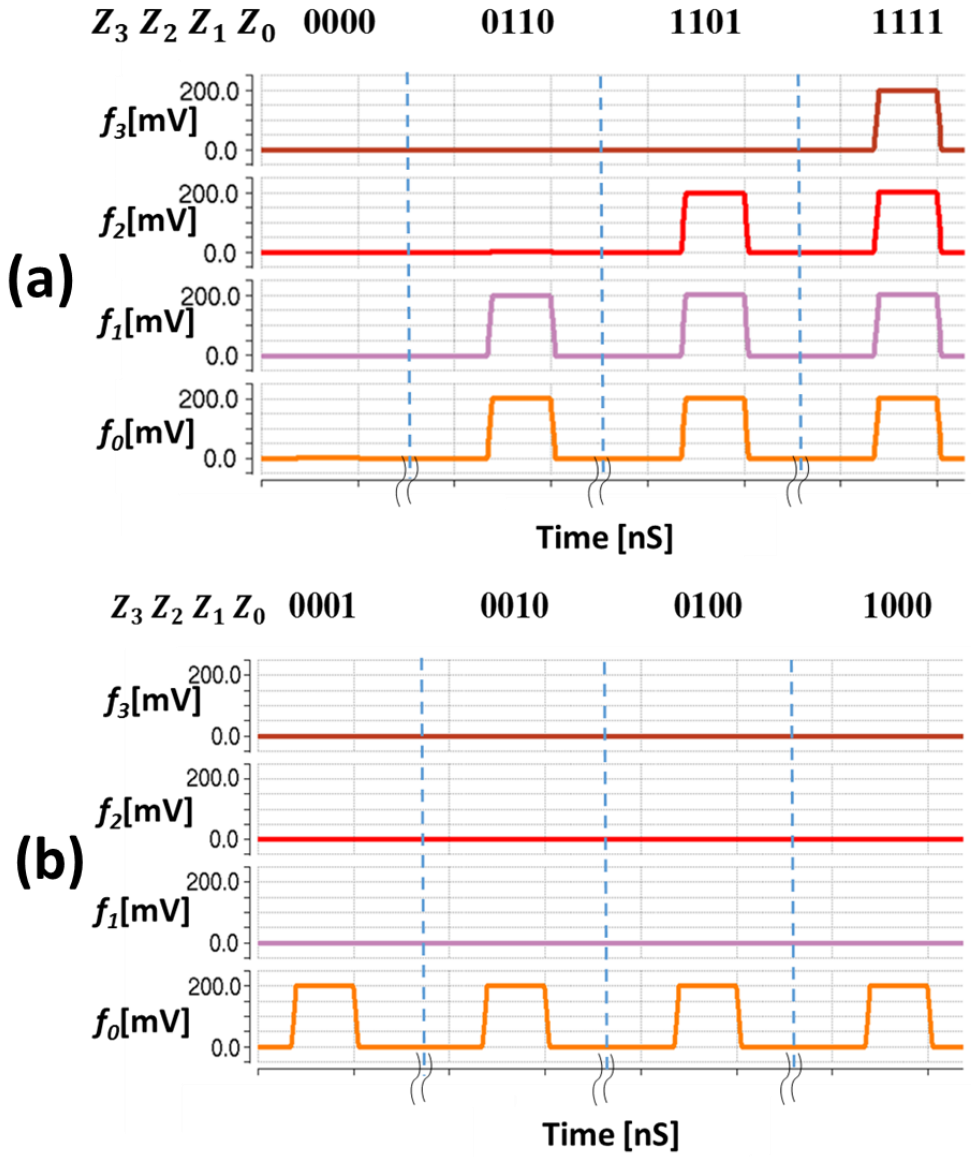


Figure 14. Simulation results of a four-bit set sort using a four by four memristive Akers array without CMOS selectors. (a) Different output values and (b) different inputs, all with a single logical one and three zeros. The output is therefore the same for all input cases. The execution voltage V_r is 200 mV.

3.3 Multistate Registers and Continuous Flow Multithreading

This secion contains the following papers:

- S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Multithreading," *IEEE Computer Architecture Letters*, 2013 (in press).
- R. Patel, S. Kvatinsky, E. G. Friedman, and A. Kolodny, "Multistate Register Based on Resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI)*, (in review).
- S. Kvatinsky, Y. H. Nacson, R. Patel, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "On the In-Die 3D Integration of Memory in CMOS Metal Layers and Its Implications on Processor Microarchitecture," submitted to *the Annual IEEE/ACM International Symposium on Microarchitecture*.

Memristor-Based Multithreading

Shahar Kvatinsky, Yuval H. Nacson, Yoav Etsion, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser

Abstract— Switch on Event Multithreading (SoE MT, also known as coarse-grained MT and block MT) processors run multiple threads on a pipeline machine, while the pipeline switches threads on stall events (*e.g.*, cache miss). The thread switch penalty is determined by the number of stages in the pipeline that are flushed of in-flight instructions. In this paper, Continuous Flow Multithreading (CFMT), a new architecture of SoE MT, is introduced. In CFMT, a multistate pipeline register (MPR) holds the microarchitectural state of multiple different threads within the execution pipeline stages, where only one thread is active at a time. The MPRs eliminate the need to flush in-flight instructions and therefore significantly improve performance. In recent years, novel memory technologies such as Resistive RAM (RRAM) and Spin Torque Transfer Magnetoresistive RAM (STT-MRAM), have been developed. All of these technologies are nonvolatile, store data as resistance, and can be described as "memristors." Memristors are power efficient, dense, and fast as compared to standard memory technologies such as SRAM, DRAM, and Flash. Memristors therefore provide the opportunity to place the MPRs physically within the pipeline stages. A performance analysis of CFMT is compared to conventional SoE MT processors, demonstrating up to a 2X performance improvement, while the operational mechanism, due to the use of memristors, is low power and low complexity as compared to conventional SoE MT processors.

Index Terms — memristor; multithreaded processors; phase change memory; RRAM, STT-MRAM.

1 Introduction

Multithreading in processors have been used to improve performance in a single core for the past two decades. One low power and low complexity multithreading technique is Switch on Event multithreading (SoE MT, also known as coarse grain multithreading and block multithreading) [1], [2], [3], [20], where a thread runs inside the pipeline until an event occurs (*e.g.*, a long latency event like a cache miss) and triggers a thread switch. The state of the replaced thread is maintained by the processor, while the long latency event is handled in the background. When a thread is switched, the in-flight instructions are flushed. The time required to refill the pipeline after a thread switch is referred to as the switch penalty. The switch penalty is usually relatively high, making SOE MT less popular than simultaneous multithreading (SMT) [18] and fine-grain multithreading (interleaved multithreading) [4]. While fine-grain MT is worthwhile only for a large number of threads, the performance of SMT is constrained in practice since the number of supported threads is limited (*e.g.*, two for Intel Sandy Bridge [5]).

In this paper, Continuous Flow Multithreading (CFMT), a novel microarchitecture, is proposed. The primary concept of CFMT is to support SoE MT for a large number of threads through the use of multistate pipeline registers (MPRs). These MPRs store the intermediate state of all instructions of inactive threads, eliminating the need to flush the pipeline on thread switches. This new machine is as simple as a regular SoE MT, and has higher energy efficiency while improving the performance as compared to regular SoE MT.

Hirst *et al.* extends the SoE MT to differential multithreading (dMT) [19], proposing up to four threads running simultaneously in a single scalar pipeline for low cost microprocessors. CFMT takes a broader view of advanced SoE MT microarchitectures. CFMT extends SoE MT by enabling the use of numerous threads using multistate pipeline registers in deep pipeline machines. CFMT is applicable to any execution event that can cause a pipeline stall.

The development of new memory technologies, such as RRAM (Resistive RAM) [6] and STT-MRAM (Spin-Transfer Torque Magnetoresistive RAM) [7], enables MPRs since these devices are located in metal layers above the logic cells and are fast, dense, and power efficient. These memory devices are referred to as memristors [8], [9].

The remainder of this paper is structured as follows: the microarchitecture of a conventional SOE MT is described and CFMT is proposed in section 2, the MPR is presented in section 3, emerging memory technologies and the basic structure of a memristor-based MPR are described in section 4, and a performance analysis for SOE MT and CFMT is presented in section 5, showing 2X theoretical performance improvements as compared to conventional SOE MT. The paper is summarized in section 6.

- *S. Kvatinsky, Y. H. Mascon, A. Kolodny, and U. C. Weiser are with the Electrical Engineering Department, Technion – Israel Institute of Technology, Haifa, Israel 32000. E-mail: skva@ tx.technion.ac.il*
- *Y. Etsion is with the Electrical Engineering and Computer Science Departments, Technion – Israel Institute of Technology, Haifa, Israel 32000.*
- *E. G. Friedman is with the Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627.*

Manuscript received 27th November 2012, manuscript accepted 13rd February 2013, and final manuscript received 18th February 2013.

DOI 10.1109/L-CA.2013.3 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

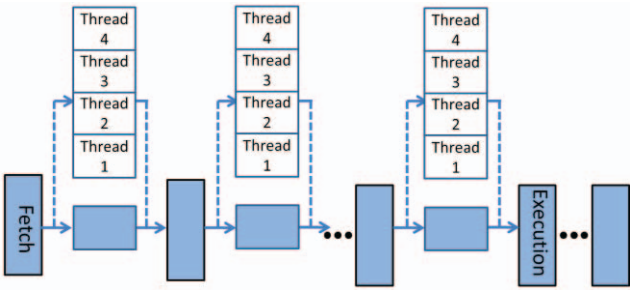


Fig. 1. Continuous Flow Multithreading (CFMT) pipeline structure. A set of multistate pipeline registers (MPRs) is located between pipeline stages. Each MPR maintains a single bit of the state of an instruction from all threads. The number of MPRs is the number of bits required to store the entire state of an instruction in the specific pipeline stage.

2 Continuous Flow Multithreading (CFMT)

To reduce the thread switch penalty, a new thread switching mechanism for SOE MT is proposed. In CFMT, pipeline registers are replaced by MPRs, as shown in Figure 1. For each pipeline stage, an MPR stores the state of the instructions from all threads. Thus, in the case of a thread switch, there is no need to flush all subsequent instructions. The processor saves the state of each instruction from the switched thread in the relevant MPR in each pipeline stage, while handling the operation of the long latency instruction in the background. Instructions from the new active thread are inserted into the pipeline from the MPR, creating a continuous flow of instructions within the pipeline. When no thread switching is required, the pipeline operates as a regular pipeline and each MPR operates as a conventional pipeline register. When the long latency instruction is completed, the result is written directly into the MPR in the background. In CFMT, the thread switch penalty is determined by the time required to change the active thread in the MPR, i.e., the time required to read the state of the new, previously inactive thread from the MPR. For a fast MPR, the thread switch penalty is significantly lower than in conventional SOE MT and the performance therefore increases significantly.

3 Multi-State Pipeline Register (MPR)

The logic structure of a multistate pipeline register (MPR) is shown in Figure 2. Each MPR stores data for multiple threads, one bit per thread. The total size of an MPR is therefore n bits, where n is the maximal number of threads. For each pipeline stage, the state of the instruction is stored in a set of MPRs with common control signals for thread management and switching. The MPR has one active thread (the current thread) for which the data can be read and written during operation of the processor, as in a regular pipeline register. During a thread switch, the active thread changes while the data of the previously active thread is maintained in the MPR. The MPR can therefore store data for all threads running in the machine. The time required to change the active thread in the MPR depends on the specific circuit structure of the MPR. This time determines the thread switch

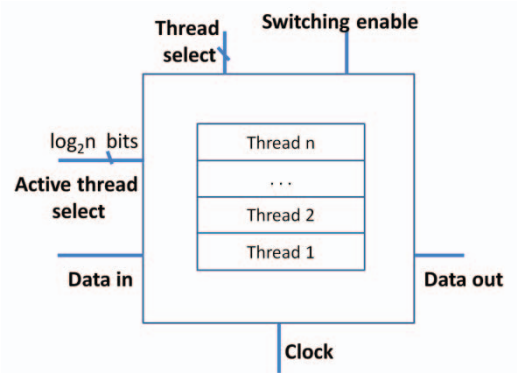


Fig. 2. The logic structure of a multistate pipeline register (MPR). An MPR maintains a single bit of the state of an instruction from all threads (stores n bits of data), where only one thread is active at a time. The MPR is synchronized by the processor clock and can switch the active thread.

penalty of CFMT. A typical thread switch penalty in CMFT is in the range of 1 to 3 clock cycles, a significant improvement as compared to SOE MT (typically 8 to 15 clock cycles).

4 Emerging Memory Technologies

Over the past decade, new technologies have been considered as potential replacements for the traditional SRAM/DRAM-based memory system to overcome scaling issues, such as greater leakage current. These emerging technologies include PCM (Phase Change Memory) [10], PMC (Programmable Metallization Cell, also known as CBRAM) [11], FeRAM (Ferroelectric RAM) [12], RRAM (Resistive RAM) [9], and STT-MRAM (Spin Transfer Torque Magnetoresistive RAM) [13].

While the physical mechanisms for these emerging memory technologies are different, all of these technologies are nonvolatile with varying resistance and can therefore be considered as memristors [8]. These emerging memory technologies are fabricated by introducing a special insulator layer between two layers of metal which can be integrated into a CMOS process, stacked vertically in multilayer metal structures physically above the active silicon transistors. This fabrication technique provides a high density of memory bits above a small area of active silicon. Memristive memory cell sizes are approximately 1 to 4 F^2 for RRAM and 8 to 45 F^2 for STT-MRAM, as compared to SRAM (60 to 175 F^2) and DRAM (4 to 15 F^2) [14], where F is the minimum feature size of the technology.

RRAM and STT-MRAM are both relatively fast [15]. STT-MRAM does not exhibit any endurance issues, while it is believed that the endurance issue of RRAM will be overcome in the near future [16]. Since memristors are dense, fast, and power efficient, these devices are attractive for use within the processor as an MPR. The basic structure for a set of memristor-based MPRs is shown in Figure 3.

For a memristor-based MPR, each thread has its own memristor-based layer, while the bottom CMOS layer is used for the active thread running within the pipeline. The bottom layer consists of standard CMOS pipeline registers, compatible with CMOS logic. During a thread

DOI 10.1109/L-CA.2013.3 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

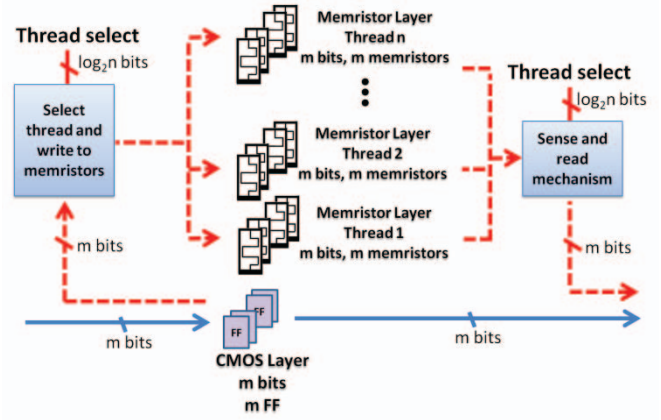


Fig. 3. Set of memristor-based multistate pipeline registers (MPRs). Each thread has its own memristor-based layer, where every bit is stored in a single memristor. The active thread is located in the bottom CMOS layer. During regular operation of the pipeline, only the CMOS layer is active (blue line) and all memristor-based layers are disabled, exploiting the nonvolatility of the memristors to save power. During a thread switch (red dashed line), the data from the CMOS layer is written into the relevant memristor-based layer, while the state of the new active thread is read and transferred to the next pipeline stage.

switch, data is copied from the CMOS layer to a specific memristor-based layer that corresponds to the previously active thread. The data from the new active thread is read into the next pipeline stage that receives the state of the new thread. When no thread switch occurs, only the bottom CMOS layer is active and the memristor layers are in standby mode. It is possible to completely disable the memristor layers and save power due to the nonvolatility of memristors.

To determine the thread switch penalty for a memristor-based MPR, only sensing the memristor layer of the new active thread is considered since the copy operation of the bottom CMOS layer to a memristor layer can be masked using buffers. This latency is determined by the read time of a memristor (sensing the data in the memristive layer). Due to the high density of memristors, the area overhead can be neglected (less than 0.1% of the pipeline area for 16 active threads [23]). This overhead is primarily due to the write mechanism and can be further optimized by separating the read and write mechanisms.

5 Performance Analysis

The performance (in CPI - cycles per instruction) of an SoE processor depends upon whether the number of threads is sufficient to overlap long latency events. Two regions of operation exist in SoE processors, depending upon the number of threads running in the machine. The *unsaturated region* is the region where the number of threads is smaller than the number required for concealing a long latency event. The behavior of the pipeline in this region is illustrated in Figure 4a. The analytic model assumes that the execution behavior in the pipeline is periodic. The period is determined by the execution of $1/r_m$ instructions from the same thread, where r_m is the average fraction of memory operations in the instruction stream. One instruction is a long latency instruction (*i.e.*,

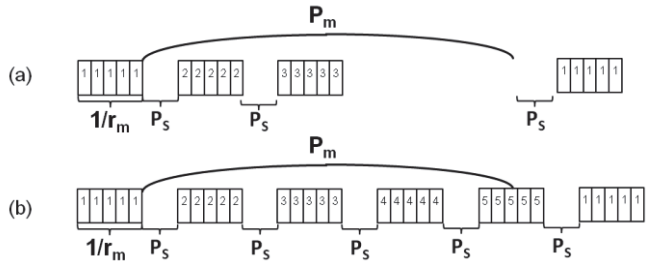


Fig. 4. The executed instructions in the two regions: (a) the unsaturated region, and (b) the saturation region. Each block is an instruction. The numbers indicate the thread number.

the instruction that triggers the thread switch; in this paper, an L1 cache miss is assumed as the trigger, with a miss penalty of P_m cycles) and the remaining instructions are low latency instructions with an average CPI of CPI_{ideal}. During execution of the long latency instruction, other instructions from different threads run within the machine. For these instructions, a periodic behavior is again assumed which also triggers a thread switch. For the unsaturated region, it is assumed that there is an insufficient number of instructions to overlap the P_m cycles required to execute the long latency instruction. The *CPI* in the unsaturated region is

$$CPI_{unsat} = \frac{CPI_{ideal} + P_m \cdot r_m \cdot MR(n)}{n}, \quad (1)$$

where n is the number of threads running in the machine and $MR(n)$ is the miss rate of the L1 cache. Note that CPI_{unsat} is limited by CPI_{sat}, as determined in (2).

When a sufficient number of threads run on the machine, the long latency instruction can be completely overlapped, and a second region, named the *saturation region*, is reached. In the saturation region, the thread switch penalty (P_s clock cycles) influences the behavior, which effectively limits the number of threads (above a specific number of threads there is no change in performance). The behavior of the pipeline in the saturation region is illustrated in Figure 4b. Assume all of the threads exhibit the same average behavior and $P_m >> CPI_{ideal}/r_m$ (*i.e.*, the miss penalty is significantly longer than the execution time of the short latency instructions). The *CPI* in the saturation region is

$$CPI_{sat} = CPI_{ideal} + P_s \cdot r_m \cdot MR(n) \quad (2)$$

In a conventional SOE MT, the switch penalty P_s is determined by the number of instructions flushed during each switch. In CFMT, however, the switch penalty is the MPR read time T_m, *i.e.*, the time required to read the state from the MPR and transfer this state to the next pipeline stage. In the case of a memristor-based MPR, the switch penalty is the time required to read the data from the memristor layer. From (2), if the value of T_m is lower than P_s, the performance of the processor in the saturation region is significantly improved, where the speedup is

$$Speedup_{sat} = 1 + \frac{r_m \cdot MR(n)}{CPI_{ideal} + T_m \cdot r_m \cdot MR(n)} \cdot (P_s - T_m). \quad (3)$$

Note that in the unsaturated region, the exact *CPI* of the CFMT is slightly better (lower) than a conventional SoE MT processor due to the improved switch penalty. The

DOI 10.1109/L-CA.2013.3 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

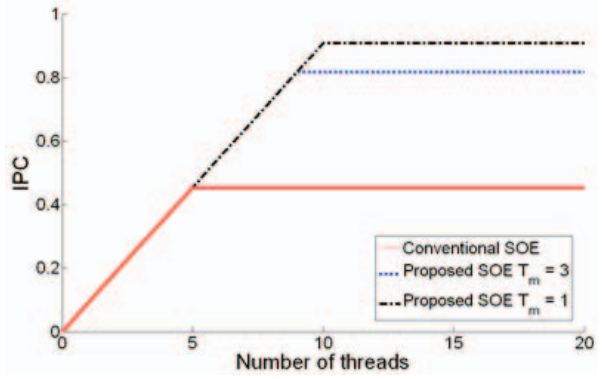


Fig. 5. The IPC of the Continuous Flow MT (CFMT) as compared to a conventional SoE MT processor (solid line). The memristor read time, which determines the thread switch penalty, is three clock cycles or one clock cycle. The IPC of CFMT is two times greater (2X improvement) than a conventional SOE MT for T_m = 1 cycle for a constant miss rate, MR = 0.25, r_m = 0.25, P_s = 20 cycles, and P_m = 200 cycles. Further reductions in P_m will linearly reduce the performance improvement (*e.g.*, for P_m = 50 cycles, the improvement in saturation performance is approximately 25%).

IPC of the proposed machine as compared to a conventional SoE machine is shown in Figure 5. The proposed machine exhibits a 2X performance improvement for a constant miss rate when operating in the saturation region. For varying miss rates (particularly with large P_m), the behavior of the *CPI* is similar to the behavior reported in [17]. Preliminary simulations have been performed on GEM5 [21], exhibiting a saturation performance improvement of approximately 50% for the SPEC MCF benchmark [22].

6 Conclusions

In this paper, a new architecture for a multithread processor, Continuous Flow Multithreading (CFMT), is proposed. This architecture is based on multi-state pipeline registers (MPR) to save the thread state in the case of an event (*e.g.*, an L1 cache miss). CFMT greatly reduces the thread switch penalty and eliminates the wasted energy of repeating instructions.

An analytic model of the performance of a conventional SoE MT and the CFMT is described. It is shown that a CFMT processor can exhibit up to a 2X performance improvement as compared to a conventional SoE MT. CFMT has a simple control mechanism and can therefore maintain more threads than modern SMT processors. The performance of the CFMT architecture is comparable to SMT processors with lower complexity and power consumption.

Emerging memristive technologies enable low power MPRs that can maintain a large number of threads in the same area of the regular pipeline registers. The memristor-based MPR demonstrates the attractiveness of memristors as a means to overcome power and performance deficiencies of existing system structures, and opens opportunities for novel processor microarchitectures.

Acknowledgments

This work was supported by the Hasso Plattner Institute. The authors thank Ravi Patel for his comments and area overhead estimation and to Nimrod Wald and Guy Satat for their help in evaluating the architecture.

References

[1] R. Gabor, S. Weiss, and A. Mendelson, "Fairness Enforcement is Switch On Event Multithreading," *ACM Transactions on Architecture and Code Optimization*, Vol. 4, No. 3, Article 15, pp. 1-34, September 2007.
[2] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel, "A Multithreaded PowerPC Processor for Commercial Servers," *IBM Journal of Research and Development*, Vol. 44, No. 6, pp. 885-898, November 2000.
[3] C. McNairy and R. Bhatia, "Montecito - The Next Product in the Itanium Processor Family," *Hot Chips* 16, August 2004.
[4] B. J. Smith, "Architecture and Applications of the HEP Multiprocessor Computer System," *Proceedings of SPIE Real Time Signal Processing IV*, pp. 241-248, 1981.
[5] L. Gwennap, "Sandy Bridge Spans Generations," *Microprocessor Report* (www.MPRonline.com), September 2010.
[6] R. Waser and M. Aono, "Nanoionics-Based Resistive Switching Memories," *Nature Materials*, Vol. 6, pp. 833-840, November 2007.
[7] Y. Huai, "Spin-Transfer Torque MRAM (STT-MRAM) Challenges and Prospects," *AAPPS Bulletin*, Vol. 18, No. 6, pp. 33-40, December 2008.
[8] L. O. Chua, "Memristor – the Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.
[9] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories – Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, Vol. 21, No. 25-26, pp. 2632-2663, July 2009.
[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 2-13, June 2009.
[11] M. N. Kozicki and W. C. West, "Programmable Metallization Cell Structure and Method of Making Same," *U. S. Patent No. 5,761,115*, June 1998.
[12] J. F. Scott and C. A. Paz de Araujo, "Ferroelectric Memories," *Science*, Vol. 246, No. 4936, pp. 1400-1405, December 1989.
[13] Z. Diao et al, "Spin-Transfer Torque Switching in Magnetic Tunnel Junctions and Spin-Transfer Torque Random Access Memory," *Journal Of Physics: Condensed Matter*, Vol. 19, No. 16, pp. 1-13, 165209, April 2007.
[14] International Technology Roadmap for Semiconductor (ITRS), 2009.
[15] A. C. Torrezan, J. P. Strachan, G. Medeiros-Riveiro, and R. S. Williams, "Sub-Nanosecond Switching of a Tantalum Oxide Memristor," *Nanotechnology*, Vol. 22, No. 48, pp. 1-7, December 2011.
[16] J. Nickel, "Memristor Materials Engineering: From Flash Replacement Towards a Universal Memory," *Proceedings of the IEEE International Electron Devices Meeting*, December 2011.
[17] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser, "Many-Core vs. Many-Thread Machines: Stay Away From the Valley," *Computer Architecture Letters*, Vol. 8, No. 1, pp. 25-28, May 2009.
[18] D. M. Tullsen, S. J. Eggers, and H. M. Levy, " Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 392-403, June 1995.
[19] J. W. Haskins, K. R. Hirst, and K. Skadron, " Inexpensive Throughput Enhancement in Small-Scale Embedded Microprocessors with Block Multithreading: Extensions, Characterization, and Tradeoffs," *Proceedings of the IEEE International Conference on Performance, Computing, and Communications*, pp. 319- 328, April 2001.
[20] M. K. Farrens and A. R. Pleszkun, "Strategies for Achieving Improved Processor Throughput," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 362-369, May 1991.
[21] http://www.m5sim.org/
[22] SPEC CPU2006 benchmark suite. http://www.spec.org/cpu2006/
[23] Private discussion with Ronny Ronen, Intel.

DOI 10.1109/L-CA.2013.3 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

Multistate Register Based on Resistive RAM

Ravi Patel, *Student Member, IEEE*, Shahar Kvatinsky, *Student Member IEEE*, Eby G. Friedman, *Fellow IEEE*, and Avinoam Kolodny, *Senior Member IEEE*

***Abstract*—In recent years, memristive technologies, such as resistive RAM (RRAM), have emerged. These technologies are usually considered as replacements to SRAM, DRAM, and Flash. In this paper, a novel digital circuit, the multistate register, is proposed. The multistate register is different than conventional types of memory, and is used to store multiple data bits, where only a single bit is active and the remaining data bits are idle. The active bit is stored within a CMOS flip flop, while the idle bits are stored in an RRAM crossbar co-located with the flip flop. It is demonstrated that additional states require an area overhead of 1.4% per state for a 64 state register. The use of multistate registers as pipeline registers is demonstrated for a novel multithreading architecture – continuous flow multithreading (CFMT), where the total area overhead in the CPU pipeline is only 2.5% for 16 threads as compared to a single thread CMOS pipeline. The use of multistate registers in the CFMT microarchitecture enables higher performance processors (40% average performance improvement) with relatively low energy (6.5% average energy reduction) and area overhead.**

Keywords—RRAM; memristor; memristive device; flip flop; multithreading

I. Introduction

Memristive technologies [1-3] have been proposed to augment existing state-of-the-art CMOS circuits. One interesting memristive technology is resistive RAM (RRAM) [4-8]. RRAM-based memories can be integrated with existing digital circuits to increase functionality and system throughput. RRAM is a two terminal device that exhibits the properties of nonvolatility and high density. Unlike charge-based memories, information in an RRAM is stored by modulating the material state. An RRAM cell dissipates no static power to store a state and provides immunity to radiation and noise induced soft errors. Fabrication of these devices generally requires deposition of a thin film material. The integration of these devices with CMOS is constrained primarily by lithographic patterning limits. Thus memristors scale with existing CMOS technologies.

The traditional approach of increasing CPU clock frequency has abated due to constraints on power consumption and density. To increase performance with each CMOS generation, thread level parallelism is exploited with multi-core processors [9]. This approach utilizes an increasing number of CMOS transistors to support additional cores on the same die, rather than increase the frequency of a single processor. This larger number of cores, however, has increased static power. Multithreading is an approach to enhance performance of an individual core by increasing logic utilization [10], without additional static power consumption. Handling each thread, however, requires duplication of resources (*e.g.*, register files, flags, pipeline registers). This added overhead increases the area, power, and complexity of the processor, potentially increasing on-chip signal delays. The thread count is therefore typically limited to two to four threads per core in modern general purpose processors [11].

The high density, nonvolatility, and soft error immunity exhibited by resistive random access memory (RRAM) enables novel tradeoffs in digital circuit design, allowing new mechanisms to increase thread count without changing the static power. These tradeoffs support innovative memory structures for novel microarchitectures. In this paper, a memristive multi-state pipeline register (MPR) is proposed that exploits these properties to enable higher throughput computing. The MPR is compatible with existing digital circuits while leveraging RRAM devices to store multiple machine states within a single register. This behavior enables an individual logic pipeline to be densely integrated with memory while retaining state information for multiple independent, on-going operations. The state information for each operation can be stored within a local memory and recalled at a later time, allowing computation to resume without flushing the pipeline.

This functionality is useful in multithreaded processors to store the state of different threads. This situation is demonstrated in the case study of a novel microarchitecture – continuous flow multithreading (CFMT) [12]. It is shown that including an RRAM MPR within the CFMT microarchitecture enhances the performance of a processor, on average, by 40%, while reducing the energy, on average, by 6.5%. The proposed

Manuscript received 17th December 2013; revised 6th May 2014; accepted xxx. This work was partially supported by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, by the Binational Science Foundation under Grant no. 2012139, by the National Science Foundation under Grant no. CCF-1329314, and by grants from Qualcomm, Cisco Systems, and Samsung.

R. Patel and E. G. Friedman are with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627, USA.

S. Kvatinsky and A. Kolodny are with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel. (S. Kvatinsky corresponding author phone: 972-77887-1923; fax: 972-4829-5757; e-mail: skva@tx.technion.ac.il).

MPR circuit can also be used as a multistate register for applications other than pipeline registers.

Background of RRAM and crosspoint style memories is reviewed in Section II. The operation of the multistate register is presented in Section III. The simulation setup and circuit evaluation process are described in Section IV. A case study examining the multistate register as a pipeline register within a CPU is presented in Section V, followed by some concluding remarks in Section VI.

II. Background

Memristors and memristor-based arrays behave differently than standard CMOS SRAM memory arrays due to the different properties of RRAM devices. The following section outlines the basic operation of memristive devices and describes memristor-based crosspoint structures.

A. Background of memristor and RRAM

Memristors [13] and memristive devices [14] behave as resistors, where the resistance is modulated by an applied bias. Positive and negative biases increase or decrease, respectively, the resistance of the device. In general, a bias applied for a longer duration produces a greater change in resistance. A larger voltage will generally increase the speed of the resistance change. The device may also exhibit a threshold voltage or current, such that the resistance will change only if the bias exceeds the threshold specific to the device technology [15-17]. Once the bias is removed, the final resistance of the memristor is retained without dissipating any power.

One interesting memristive technology is RRAM, where oxide-based materials (*e.g.*, TaO, TiO, SiO) [18, 19] rely on the migration of dopants to switch the resistance of a tunnel barrier. Dopant chains form through the oxide and reduce the thickness of the tunneling gap. An increase in the gap thickness gives rise to an increase in the resistance of the device while a decrease reduces the resistance. Currently, RRAM is considered a good candidate to replace Flash memory and is being widely investigated both in industry and academia.

The exact behavior of RRAM devices varies for different oxide materials. To simulate the behavior of memristive circuits, a general device model is used – the TEAM model [20]. In the TEAM model, the behavior of the resistive device is represented by the following expressions,

$$\frac{dx(t)}{dt} = \begin{cases} k_{off}\cdot\left(\dfrac{i(t)}{i_{off}}-1\right)^{\alpha_{off}}\cdot f_{off}(x), & 0 < i_{off} < i, \quad (1a) \\ 0, & i_{on} < i < i_{off}, \quad (1b) \\ k_{on}\cdot\left(\dfrac{i(t)}{i_{on}}-1\right)^{\alpha_{on}}\cdot f_{on}(x), & i < i_{on} < 0, \quad (1c) \end{cases}$$

$$v(t) = \left[R_{ON} + \frac{R_{OFF}-R_{ON}}{x_{off}-x_{on}}\left(x - x_{on}\right)\right]\cdot i(t), \quad (2)$$

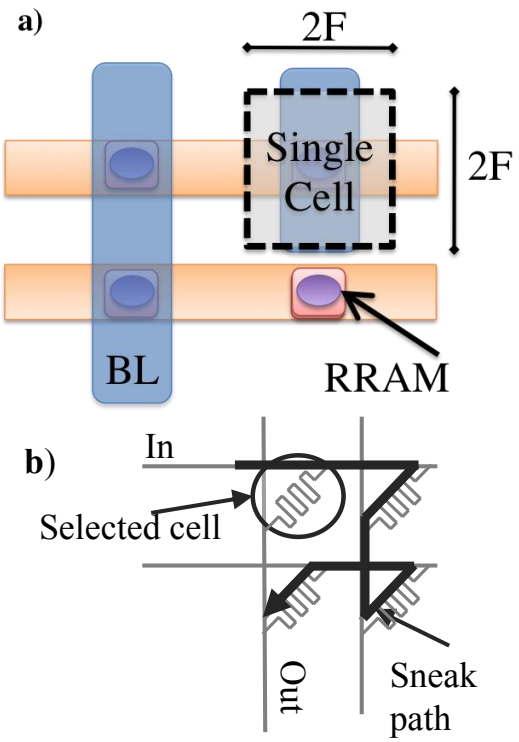


Figure 1. RRAM crosspoint (a) structure, and (b) an example of a parasitic sneak path within a 2 x 2 crosspoint array.

where k_{off} and k_{on} are fitting parameters, α_{on} and α_{off} are adaptive nonlinearity parameters, i_{off} and i_{on} are current threshold parameters, $f_{on}(x)$ and $f_{off}(x)$ are window functions, R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, and x_{on} and x_{off} are, respectively, the minimum and maximum allowed value of the internal state variable x. The window function returns a value between zero and one and describes the rate at which the change of the state variable becomes nonlinear near the minimum and maximum resistance of a memristor. A Joglekar window function is used with a p-coefficient of two [21]. An I-V curve of a memristive device based on the TEAM model is shown in Figure 2a, exhibiting a pinched hysteresis loop.

B. Crosspoints and nonlinearity

RRAM has the greatest density when utilized in a crosspoint configuration. In this structure, a thin film is sandwiched between two sets of parallel interconnects. Each set of interconnects is orthogonal, allowing any individual memristive device to be selected by biasing one vertical and one horizontal metal line. In this configuration, the circuit density is only limited by the available metal pitch. The structure of a crosspoint is shown in Figure 1a.

Crosspoint arrays have the inherant problem of sneak path currents where currents propagate between the two selected lines through unselected memristors. The sneak path phenomenon is illustrated in Figure 1b. The nonlinear I-V characteristic of certain memristive devices lessens the sneak path phenomenon [22]. This nonlinearity can be achieved by

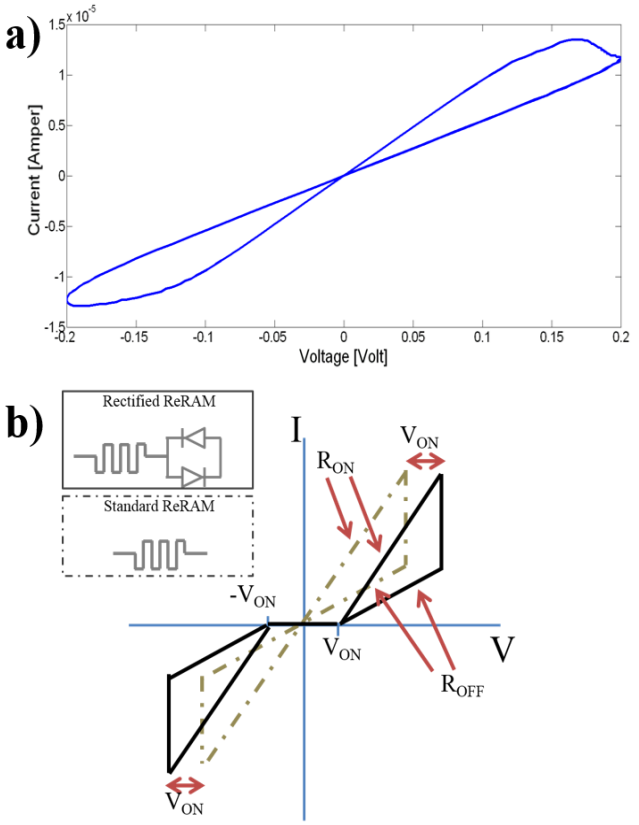


Figure 2. I-V characteristic of a memristor for (a) a ThrEshold Adaptive Memristor (TEAM) model with a 0.2 volt sinusoidal input operating at a frequency of 2 GHz, and (b) resistive devices with and without ideal cross-coupled diodes. The parameters of the TEAM models are listed in Table II. V_{ON} is the on-voltage of the diode, and R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor.

depositing additional materials above or below the memristive thin film. Depending on the material system used for RRAM, the nonlinearity can result from an insulator to metal transition or a negative differential resistance [22]. From a circuits perspective, the combined device can be modeled as a pair of cross coupled diodes in series with a memristor, as shown in the inset of Figure 2b. Since the rectifying structure requires an additional thin film layer, there is no effect on the area of the crosspoint structure.

An I-V curve of a memristive device with cross coupled diodes is shown in Figure 2b. The high resistance of the unselected devices reduces sneak currents and ensures that the leakage power of the array is relatively small. Reducing sneak currents ensures that the leakage power of the array is relatively small. A DC analysis of the crosspoint on and off currents is listed in Table I, where a 4 x 4 crosspoint array with RRAM devices is DC biased at 0.8 volts. These RRAM devices exhibit an on/off current ratio of 30. In an unrectified crosspoint, the observed current ratio drops to less than two. The rectified crosspoint displays a current ratio of 28.5, only 5% less than the ideal ratio of an RRAM device. Furthermore, the total power consumption is reduced by almost an order of magnitude.

TABLE I. COMPARISON OF DC ON/OFF CURRENT FOR 4 X 4 CROSSPOINT ARRAY

	I_{on} [mA]	I_{off} [mA]	Ratio	Average Active Power [mW]
Unrectified	2.3	0.132	1.7	1.45
Rectified	0.486	0.017	28.5	0.201

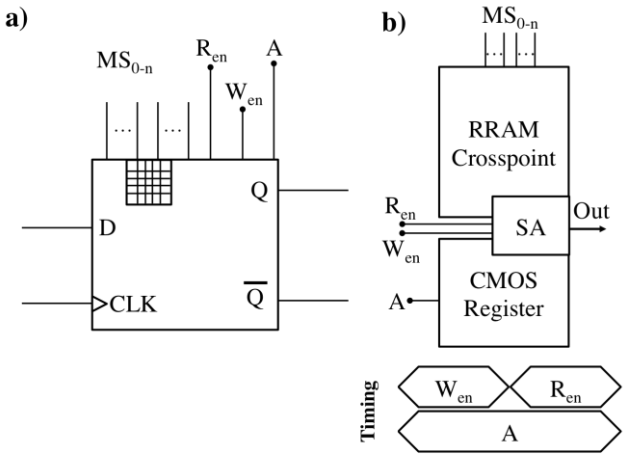


Figure 3. Multistate register element. (a) Symbol of the multistate register, and (b) block diagram with control signal timing. The symbol is similar to a standard CMOS D flip flop with the addition of a crosspoint array symbol.

III. RRAM MULTISTATE REGISTER

The multistate register is a novel circuit used to store multiple bits within a single logic gate. The multistate register is "drop-in" compatible with existing CMOS based flip flops. The element utilizes a clocked CMOS register augmented by additional sense circuitry (SA) and global memristor select (MS) lines. The symbol and topology of the multistate register are shown in Figure 3. Multistate registers can be used as pipeline registers within a processor pipeline, as shown in Figure 4 and further explained in Section V.

The MS lines select individual RRAM devices within the crosspoint memory co-located with the CMOS register. A schematic of the proposed RRAM multistate register is shown in Figure 5a. The signals W_{en} and R_{en} are global control signals that, respectively, write and read within the local crosspoint memory. Signal A sets the CMOS register into an intermediate state that facilitates writes and reads from the crosspoint. An individual RRAM device is selected using a set of global MS lines. Local writes to the RRAM crosspoint are controlled by the master stage within the register. The gates within the slave stage of the CMOS register are reconfigured to provide a built-in sense amplifier to read the RRAM crosspoint array [23]. The overhead of the additional circuitry (shown in Figure 3) is relatively small (see Section IV.B).

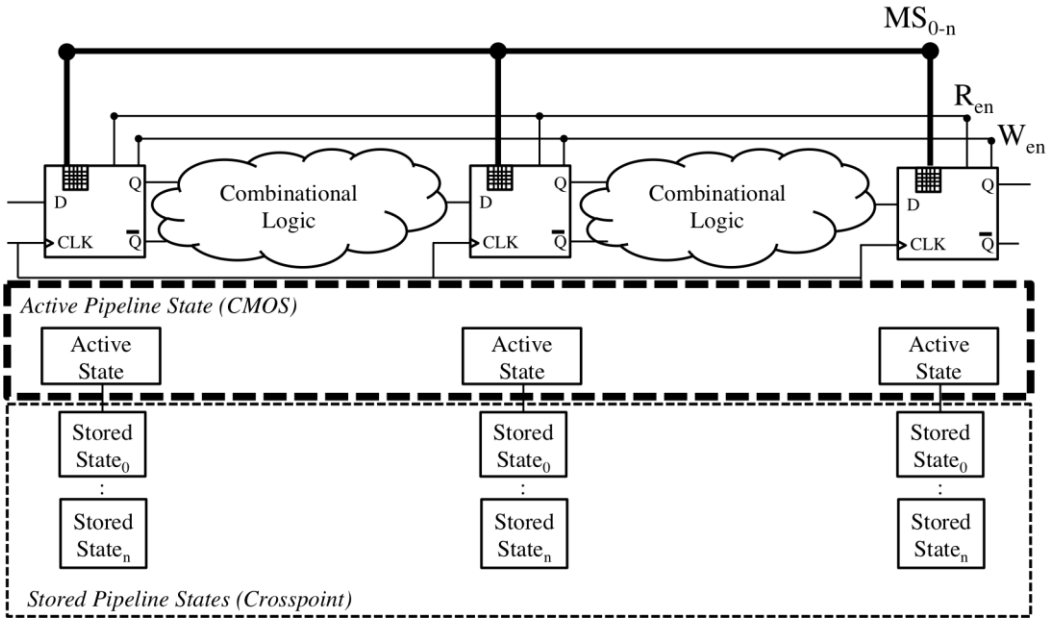


Figure 4. Multistate pipeline register (MPR) based pipeline and logic diagram of active and stored pipeline states. The MPR replaces a conventional pipeline register and time multiplexes the stored states.

The multistate register primarily operates as a CMOS register. In this mode, the structure behaves as a standard D flip flop, where a single bit is stored and is active while the idle states are stored within the RRAM crosspoint array. When global control circuitry triggers a change of the pipeline state (*e.g.*, for a pipeline stall or context switch), the circuit stores the current bit of the register and reads out the value of the next active bit from the internal RRAM-based storage. Switching between active bits consists of two phases. In the first half of the cycle, an RRAM write operation stores the current state of the register. During a write operation, the transmission gate *A* disconnects the first stage from the following stage, isolating the structure into two latches. The input latch stores the currently evaluated state, while the output latch stores the data of the previous state. Once W_{en} goes high, the input latch drives a pair of multiplexors that write the currently stored state into the RRAM cell selected by the global MS lines. The active devices during the write phase are shown in Figure 5b. The write phase may require more than half a cycle depending upon the switching time of the RRAM technology. During the second half of the clock cycle, the new active bit is selected within the resistive crosspoint array and sensed by the output stage of the CMOS D flip flop. During a read operation, the globally selected row is grounded through the common node N_{in}. The voltage on the common line N_{out} is set by the state of the RRAM cell. To bias the RRAM cell, the common line is connected through a PMOS transistor to the supply voltage V_{DD}. The voltage is sensed at the output of M_1. If R_{en} is set high, M_1 to M_5 reconfigure the last inverter stage as a single ended sense amplifier [12], and the crosspoint array is read. The active devices during the read phase are shown in Figure 5c.

The physical design of the multistate register can be achieved by two approaches. RRAM devices can be integrated between the first two metals, as illustrated in Figure 6a, or the RRAM can be integrated on the middle level metal layers, as shown in Figure 6b. The middle metal layer approach allows the RRAM to be integrated above the CMOS circuitry, saving area. A standard cell floorplan is shown in Figure 7b, where a dedicated track is provided for the RRAM interface circuitry. This dedicated track runs parallel to the CMOS track. The addition of this track wastes area in those cases where multistate registers are sparsely located among the CMOS gates. Additional routing overhead increases the area required to pass signals around the crosspoint array.

The approach illustrated in Figure 7a, where the RRAM is integrated on the lower metal layers, requires slightly more area but is compatible with standard cell CMOS layout rules. Fabrication on the lower levels maintains standard routing conventions, where the lower metal layers are dedicated to routing within the gates, and the middle metal layers are used to route among the gates.

IV. Simulation Setup and Circuit Evaluation

The multistate register has been evaluated for use within a high performance microprocessor pipeline. The latency, energy, and area of the register are described in this section as well as the sensitivity to process variations.

A. Latency and energy

The latency and energy of an MPR are dependent on the parameters of an RRAM device and the CMOS sensing circuitry built into the MPR. The RRAM device is modeled using the TEAM model [20] based on the parameters listed in Table II. The parameters of the resistive device are chosen to

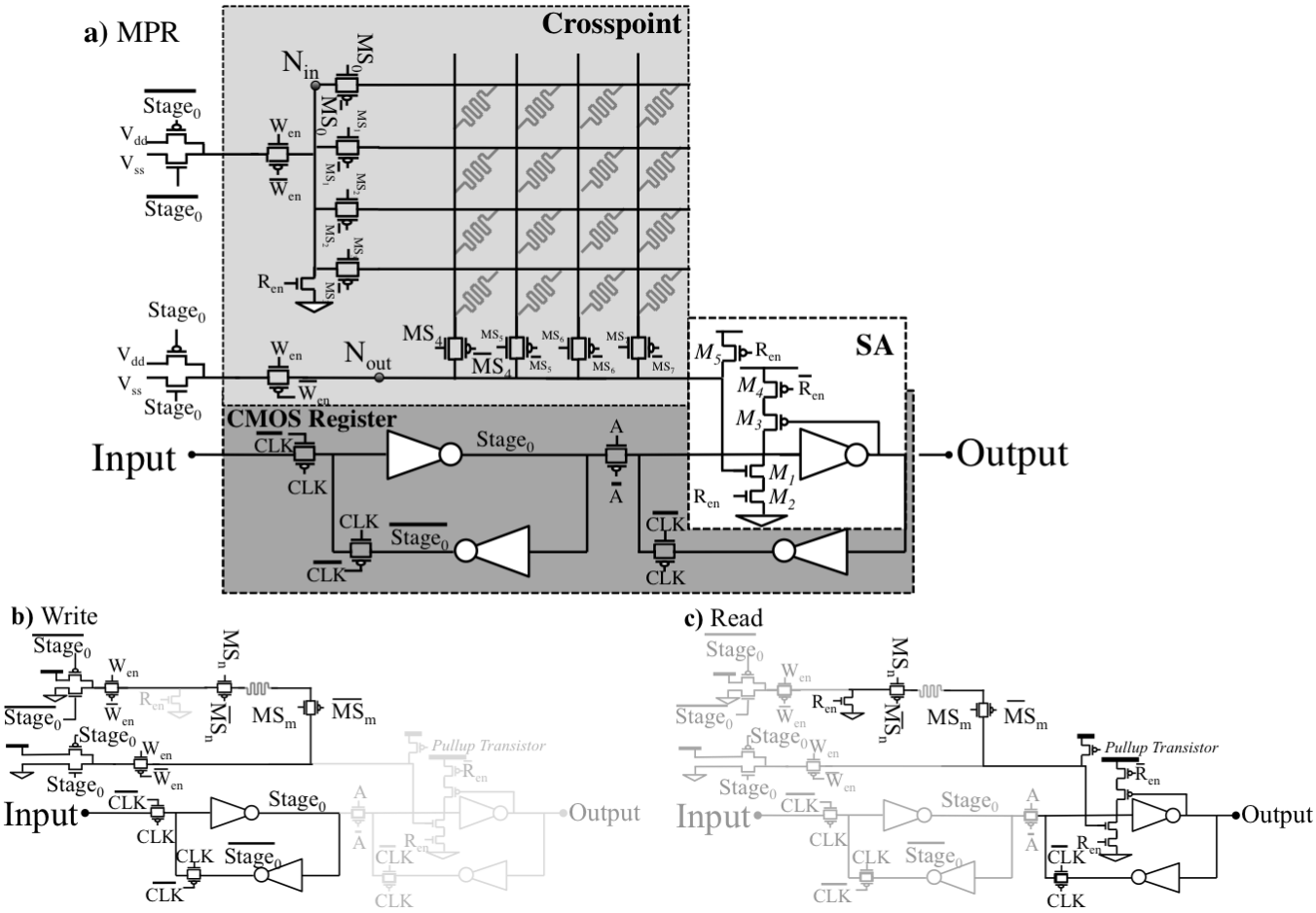


Figure 5. Proposed RRAM multistate pipeline register. (a) The complete circuit consists of a RRAM-based crosspoint array above a CMOS-based flip flop, where the second stage (the slave) also behaves as a sense amplifier. The (b) write and (c) read operations of the proposed circuit.

incorporate device nonlinearity into the I-V characteristic, as shown in Figure 2b and described in Section IIA. The multistate register is evaluated across a range of internal cross point sizes (*e.g.*, different number of states per register). The resistance of the device is extracted from [22]. The transistor and cell track sizing information is from the FREEPDK45 Standard Cell Library [24] and scaled to a 22 nm technology. Circuit simulations utilize the 22 nm PTM CMOS transistor model [25]. The RRAM and diode device parameters are listed in Table II. Standard CMOS timing information for the register is listed in Table III. The read operation requires 28.6 ps, equivalent to a 16 GHz clock frequency (the read operation is less than half a clock cycle). The register operates primarily as a CMOS register and only accesses the RRAM crosspoint array to switch between idle and active pipelines states. Note that the eight row by eight column crosspoint array is small as compared to large scale memory crosspoint arrays, and therefore places a small electrical load on the sensing circuitry. Hence, the read operation is relatively fast and does not limit the operation of the multistate register.

The performance of the multistate register is limited by the switching characteristics of the RRAM device. To maintain high performance, the desired RRAM devices must be relatively fast [29]. These characteristics are chosen to achieve a target write latency of a 3 GHz CPU. As mentioned in Section II, the RRAM write operation occurs sequentially prior to the read operation. Due to the sequential nature of the multistate register access to the RRAM array, a half cycle is devoted to the read operation.

The energy of the multistate register depends upon the RRAM switching latency, as listed in Table IV. $E_{Low\text{-}High}$ and $E_{High\text{-}Low}$ are the energy required to switch, respectively, to R_{off} and R_{on} for a single device write to the multistate register crosspoint array. Since the switching time of the memristor dominates the delay of a write to the multistate register, $E_{Low\text{-}High}$ and $E_{High\text{-}Low}$ increase linearly as the switching time increases. Note that the read energy only depends on R_{ON} and R_{OFF} and is therefore constant for different switching times. The read energy, however, depends on the size of the crosspoint (*i.e.*, the number of RRAM devices), as listed in Table V.

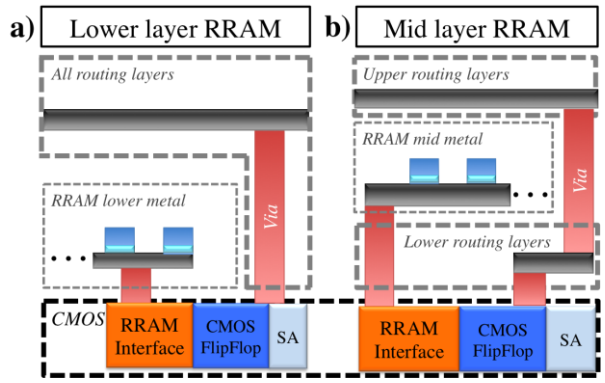


Figure 6. Vertical layout of RRAM in MPR circuit for a) lower level, and b) mid-layer crosspoint RRAM array.

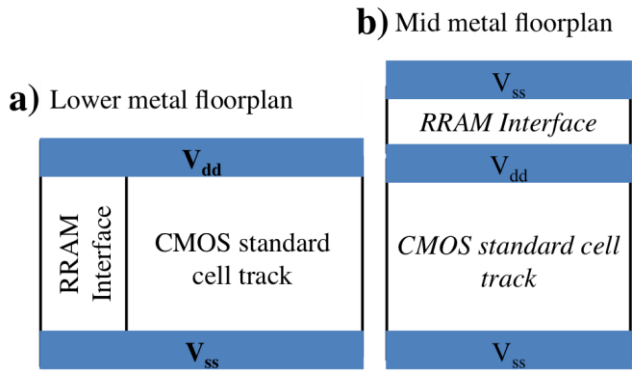


Figure 7. Planar floorplan of MPR with lower metal and upper metal RRAM layers. The RRAM array is not marked in this figure since it is located above the CMOS layer and has a smaller area footprint.

TABLE II. MEMRISTOR AND DIODE PARAMETERS

R_{on} [kΩ]	0.5
R_{off} [kΩ]	30
k_{on}	-0.021-0.07
k_{off}	0.0021-0.007
$\alpha_{on.off}$	3
i_{on} [μA]	-1
i_{off} [μA]	1
V_{ON} (diode) [V]	0.5
R_{out} (diode) [Ω]	1

TABLE III. ACCESS LATENCY OF A 16 BIT MPR

Clock to Q [ps]	11.2
Setup Time [ps]	13.2
RRAM Read [ps]	28.6

TABLE IV. WRITE LATENCY AND ENERGY OF A 16-BIT MULTISTATE REGISTER

Write Time [cycles @ 3 GHz]	0.5	1.5	2.5	3.5	4.5
$E_{Low\text{-}High}$ [fJ]	2.24	5.26	8.3	10.49	13.23
$E_{High\text{-}Low}$ [fJ]	3.78	10.33	16.89	23.5	30.08

TABLE V. READ ACCESS ENERGY OF RRAM

States per Mutistate Register	4 States	16 States	64 States
$E_{read,Off}$ [fJ]	1.6	2.2	3.5
$E_{read,On}$ [fJ]	0.33	0.41	0.71

B. Layout and physical area

The energy and latency of an MPR are dependent on both the parameters of an RRAM device and on the CMOS sensing circuitry built into the MPR. An individual crosspoint RRAM cell is 0.001934 μm^2 ($4F^2$, where F is the feature size). The layout of the proposed RRAM multistate register is shown in Figure 8. The layout of the multistate register is based on 45 nm design rules and scaled to the target technology of 22 nm. The number of RRAM devices within a crosspoint array is scaled from four devices to 64 devices. The MPR is evaluated for both the middle metal and lower metal approaches, as described in Section III. The physical area is listed in Table VI.

The transistors required to access the crosspoint, as shown in Figure 8, dominate the area overhead of both the lower metal and middle metal multistate register. Due to the relatively small on-resistance of the RRAM devices, the access transistor needs to be sufficiently large to facilitate a write operation. Additionally, CMOS transmission gates are used to ensure that there is no threshold drop across the pass transistors. As a result, the area of the crosspoint memory is only a small fraction of the area overhead of the multistate register. Note that alternative RRAM technologies with a higher R_{on} supports smaller transistors and reduced area. Under these constraints, the most area efficient structure is a 64 bit array, as the overhead per state is, respectively, 0.08 μm^2 for the lower metal approach and 3.75 μm^2 for the middle metal approach.

As shown, the middle metal register requires less area than a lower metal multistate register. As described in Section III and depicted in Figure 8b, the middle metal register requires an additional track dedicated to the control transistors within the crosspoint array. Positioning the crosspoint array over the register also adds complexity as the upper metal layers can no longer be used to route signals above the multistate register.

C. Sensitivity and device variations

The built-in sense amplifier circuit senses the RRAM based on a threshold voltage. Any voltage above the threshold of the registers produces a logical zero at the output, and any voltage below the threshold produces a logical one. Similar to digital CMOS circuits, the structure is tolerant to variability in the RRAM resistance. To evaluate the sensitivity of the circuit

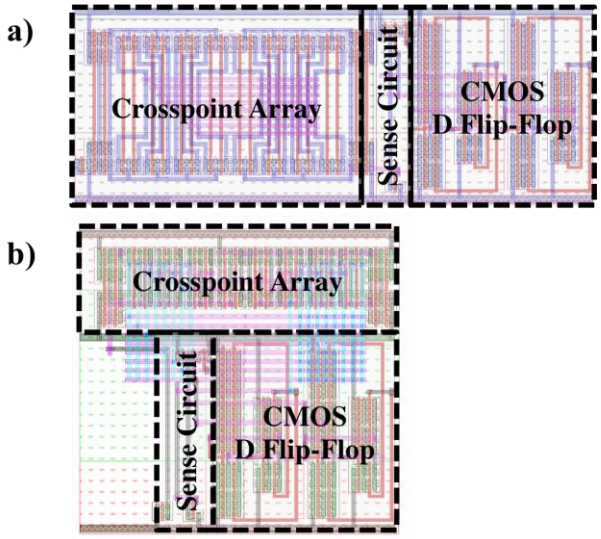


Figure 8. Physical layout of 64 state MPR within the crosspoint array on (a) lower metal layers (M1 and M2), and (b) upper metal layers (M2 and M3) above the D flip flop.

TABLE VI. MPR AREA

		Area [μm^2]	Overhead [%]	Overhead per State [%]
	CMOS Register (1 state)	2.8	-	-
Lower Metal	MPR 4 states	5.5	96.2%	24%
	MPR 16 states	6.3	126.5%	8%
	MPR 64 states	8.1	192.5%	3%
Middle Metal	MPR 4 states	3.9	39.3%	9.8%
	MPR 16 states	4.3	53.6%	3.3%
	MPR 64 states	5.2	85.7%	1.3%

to variations, the nominal R_{on} is varied from 0.35 to 0.65 kΩ. This range produces a maximum and minimum change of ±2 mV in the voltage input of the sense amplifier. For 21 kΩ > R_{off} < 39 kΩ, a voltage ranging from -40 mV to +26 mV is produced. Both ranges represent a 30% variation in the device resistance of R_{on} and R_{off}. In these cases, the correct output state is read out, indicating a high degree of tolerance to variations in the RRAM resistance.

The RRAM circuit can tolerate an R_{on} of up to 12 kΩ before the circuit produces an incorrect output. In a 64 bit multistate register; this behavior corresponds to an increase in the RRAM read delay from 78 ps to 476 ps. With increasing R_{on}, the sense amplifier no longer generates a full range signal at the output, dissipating static energy. Much of this increased delay is due to the device operating near the switching threshold of the sense amplifier.

As R_{off} varies from 30 kΩ to 300 MΩ, the performance of the circuit improves due to two effects. As the resistance increases, the voltage at the sense amplifier input also increases, placing the transistor into a higher bias state, which lowers the delay of the sense amplifier. Additionally, the large resistance of the sensed RRAM device prevents the sense line within the crosspoint array from dissipating charge, maintaining a high voltage at the input of the sense amplifier. Counterintuitively, this effect lowers the delay when R_{off} is greater than 30 MΩ. Due to the interplay of R_{on} and R_{off}, a

TABLE VII. SOE MT AND CFMT PROCESSOR CONFIGURATIONS

	Switch on Event	RRAM-based CFMT
Number of pipeline stages	10	
CMOS process	22 nm	
Clock frequency [GHz]	3	
Switch penalty [cycles]	7	1 to 5
L1 read/write latency [cycles]	0	
L1 miss penalty [cycles]	200	
Data L1 cache configuration	32 kB, 4 way set associative	
Instruction L1 cache configuration	32 kB, 4 way set associative	
Branch predictor	Tournament , lshare 18kB/gshare 8kB	

TABLE VIII. PERFORMANCE SPEEDUP FOR DIFFERENT MPR WRITE LATENCIES AS COMPARED TO SWITCH-ON-EVENT MULTITHREADING PROCESSOR FOR CPU SPEC 2006

Benchmark	MPR Write Latency [clock cycles]				
	1	2	3	4	5
libquantum	1.35	1.28	1.21	1.15	1.09
bwaves	1.22	1.15	1.08	1.04	1
milc	1.47	1.26	1.18	1.11	1.06
zeusmp	1.85	1.59	1.40	1.29	1.21
gromacs	1.53	1.32	1.21	1.17	1.14
leslie3d	1.67	1.48	1.33	1.22	1.15
namd	1.40	1.24	1.15	1.08	1.04
soplex.pds-50	1.35	1.28	1.21	1.16	1.1
lbm	1.5	1.31	1.2	1.12	1.08
bzip2.combined	1.13	1.1	1.08	1.05	1.03
gcc.166	1.35	1.28	1.21	1.15	1.09
gobmk.trevorc	1.3	1.24	1.19	1.14	1.09
h264ref.foreman_baseline	1.06	1.02	1	1	1
GemsFDTD	1.45	1.3	1.18	1.08	1.04
hmmer.nph3	1.18	1.14	1.11	1.07	1.04
soplex.ref	1.7	1.42	1.29	1.19	1.1
gcc.c-typeck	1.33	1.26	1.21	1.15	1.1
gobmk.trevord	1.29	1.23	1.18	1.13	1.08
Average	1.40	1.27	1.19	1.13	1.08

TABLE IX. ENERGY AND AREA EVALUATION FOR CFMT TEST CASE

	Switch on Event	RRAM-based CFMT	Difference
Thread switch energy [pJ]	109.9	9,1 @ 1 cycle penalty	-91.7%
		19.1 @ 2 cycle penalty	-82.6%
		29.2 @ 3 cycle penalty	-73.4%
		38.4 @ 4 cycle penalty	-65.1%
		48.2 @ 5 cycle penalty	-56.1%
Processor area [mm^2]	123.276	126.426	2.55%

TABLE X. ENERGY PER INSTRUCTION FOR VARIOUS CPU SPEC 2006 BENCHMARK APPLICATIONS

Benchmark	SoE MT [pJ/inst.]	CFMT RRAM MPR – various thread switch latencies				
		1 cycle [pJ/inst.]	2 cycles [pJ/inst.]	3 cycles [pJ/inst.]	4 cycles [pJ/inst.]	5 cycles [pJ/inst.]
libquantum	15.17	14.12	14.29	14.46	14.63	14.80
bwaves	19.63	18.83	19.03	19.25	19.42	19.42
milc	24.51	22.61	23.23	23.47	23.74	24.11
zeusmp	21.10	18.04	18.62	19.19	19.18	19.95
gromacs	30.16	27.94	28.62	29.05	29.23	29.34
leslie3d	27.27	24.72	25.20	25.68	26.08	26.39
namd	22.90	21.42	21.91	22.21	22.50	22.65
soplex.pds-50	17.62	16.52	16.71	16.88	17.03	17.20
lbm	22.54	20.29	20.90	21.36	21.76	21.94
bzip2.combined	21.86	21.44	21.51	21.65	21.65	21.72
gcc.166	19.37	18.32	18.49	18.66	18.83	19.01
gobmk.trevorc	23.05	22.15	22.28	22.71	22.56	22.71
h264ref.foreman_baseline	25.95	25.27	25.35	25.50	25.69	25.76
GemsFDTD	23.89	21.88	22.43	22.99	23.36	23.49
hmmer.nph3	24.27	23.65	23.75	23.84	23.84	24.04
soplex.ref	21.92	19.47	20.04	20.44	20.80	21.17
gcc.c-typeck	19.94	19.16	19.12	19.27	19.43	19.58
gobmk.trevord	22.73	21.71	21.87	22.40	22.25	22.40
Average	22.44	20.97	21.30	21.61	21.78	21.98

delay tradeoff therefore exists between the average resistance of the RRAM technology and the resistive ratio of the device.

The gain and offset of the sense amplifier have a small effect on the circuit performance. A higher sense amplifier gain improves the tolerance of the sense circuit to variations of the RRAM device. An offset voltage shifts the reference threshold voltage, but must be comparable to the supply voltage (0.3VDD or more) before the circuit performance is affected.

V. MULTISTATE REGISTERS AS MULTISTATE PIPELINE REGISTER FOR MULTITHREAD PROCESSORS – A TEST CASE

Replacing CMOS memory (*e.g.*, register file and caches) with non-volatile memristors significantly reduces power consumption. Multithreaded machines can exploit the high density and CMOS compatibility of memristors to store the state of the in-flight instructions within a CPU with fine granularity. Hence, using memristive technology can dramatically increase the number of threads running within a single core. This approach is demonstrated in this test case, where RRAM multistate registers store the state of multiple threads within a CPU pipeline.

In continuous flow multithreading [12], the multistate registers are used as MPRs to store the state of multiple threads. A single thread is active within the pipeline and the instructions from the other threads are stored within the MPRs. The MPRs therefore eliminate the need to flush instructions within the pipeline, significantly improving the performance of the processor, as illustrated in Figure 9.

To exemplify this behavior, the performance and energy of a CFMT processor with the proposed RRAM-based MPRs have been evaluated [26]. To evaluate the performance, the GEM5 simulator [27] is extended to support CFMT. The energy has been evaluated by the McPAT simulator [28]. The simulated processor is a ten stage single scalar ARM processor, where the execution stage operates at the eighth stage. The performance and energy of the CFMT processor are compared to a switch-on-event (SoE) multithreading processor [30], where a thread switch occurs for each long latency instruction (*e.g.*, L1 cache miss, floating point instructions), causing the pipeline to flush. The characteristics of the evaluated processors are listed in Table VII. The energy is compared to a 16 thread processor (*i.e.*, with an MPR storing 16 states) which is a sufficient number of threads to achieve the maximum performance for most benchmark applications.

The performance of the processors is measured by the average number of instructions per clock cycle (IPC), as listed in Table VIII. The average speedup in performance is 40%. A comparison of the thread switch energy is listed in Table IX. The average energy per instruction for various CPU SPEC 2006 benchmarks is listed in Table X, where the average reduction in energy is 6.5%. The area overhead for a 16 thread CFMT as compared to an SoE is approximately 2.5%, as listed in Table IX.

For the CFMT configuration described herein, the simulations show that 16 threads are sufficient to achieve the maximum performance for the vast majority of SPEC CPU 2006 benchmarks. Alternate configurations with many long latency events or different machines may benefit from additional states.

Physically, a linear increase in the number of rows and columns within the crosspoint array generates a quadratically increasing number of states and physical area, increasing the efficiency of the crosspoint array. A small increase in the number of rows and columns supports many more threads.

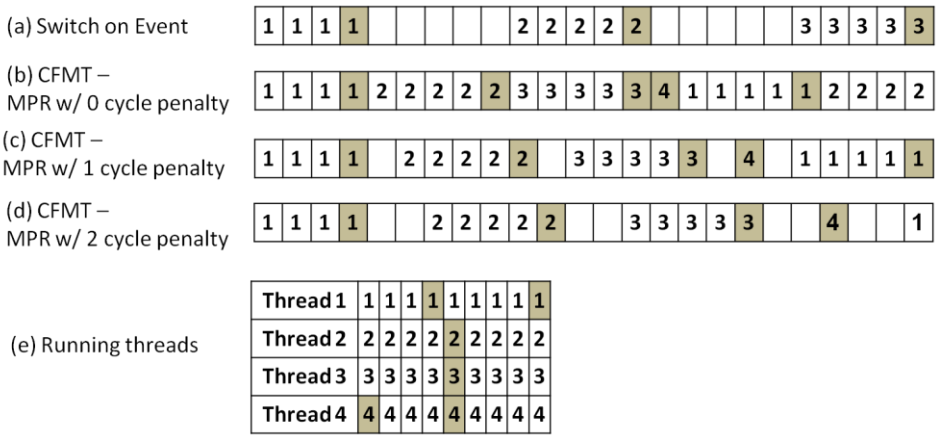


Figure 9. Illustration of different multithreading techniques with four threads (marked by the number). All processors run the same four threads as shown in (e). The latency of the 'white' and 'shaded' instructions is, respectively, a single clock cycle and ten clock cycles. For a (a) switch-on-event multithreaded processor, the long latency instruction that triggers a thread switch is the shaded instruction and the thread switch penalty is five clock cycles due to the pipeline flush. For continuous flow multithreading, the thread switch penalty depends upon the read and write times of the multistate register, and is lower than traditional switch-on-event processors. The different thread switch penalties illustrated in this example are (b) zero for an ideal multistate register, (c) one clock cycle, and (d) two clock cycles. The performance (measured by the instructions per cycles) in this example is (a) 7/12, (b) 1 (71% improvement as compared to switch-on-event), (c) 0.83 (43% improvement), and (d) 0.67 (14% improvement) [12].

However, as previously mentioned, 64 states is sufficient for most applications.

For the MPR to enhance performance, the cost of a thread switch must be smaller than the latency of a cache miss or other long latency events. This situation is typical for all practical thread switching events.

VI. Conclusions

Emerging memory technologies, such as RRAM, are more than just a drop-in replacement to existing memory technologies. In this paper, a RRAM based multistate register is proposed using an embedded array of memristive memory cells within a single flip flop. The multistate register can be used to store additional data that is not conventionally contained within the computational pipeline.

The proposed multistate register is relatively fast due to the physical closeness of the CMOS and RRAM devices. A 16 state multistate register requires only 54% additional area as compared to a single state standard register. The multistate register is also relatively low power due to the non-volatility of the resistive devices.

As an example, the proposed multistate register has been applied to a continuous flow multithreading processor, exhibiting a significant performance improvement of 40% with low energy as compared to a conventional switch-on-event processor. An RRAM-based MPR therefore enables novel microarchitectures, such as the CFMT. The proposed multistate register is shown to significantly improve performance and reduce energy with a small area overhead.

Acknowledgments

The authors thank Yoav Etsion, Yuval H. Nacson, and Uri C. Weiser for their contributions.

References

[1] D. B. Strukov, G. S.Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, Vol. 453, pp. 80-83, May 2008.

[2] M. Hosomi et al., "A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM," *Proceedings of the IEEE International Electron Devices Meeting*, pp. 459–462, December 2005.

[3] L. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A*, Vol. 102, No. 4, pp. 765-783, March 2011.

[4] H. S. Wong *et al,* "Metal–Oxide RRAM," *Proceedings of the IEEE*, Vol. 100, No. 6, pp. 1951-1970, June 2012.

[5] Y. Ho, G. M. Huang, and P. Li. "Nonvolatile Memristor Memory: Device Characteristics and Design Implications," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 485-490, November 2009.

[6] D. R. Lamb and P. C. Rundle, "A Non-Filamentary Switching Action in Thermally Grown Silicon Dioxide Films," *British Journal of Applied Physics*, Vol. 18, No.1, pp. 29-32, January 1967.

[7] G. M. Ribeiro *et al.*, "Designing Memristors: Physics, Materials Science and Engineering," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 2513–2516, May 2012.

[8] J. J. Yang *et al.*, "Memristive Switching Mechanism for Metal/Oxide/Metal Nanodevices," *Nature Nanotechnology*, Vol. 3, No. 7, pp. 429–433, June 2008.

[9] J. Li and J. F. Martinez, "Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors," *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp.124-134, March 2005.

[10] D. M. Tullsen, S. J. Eggers, and H. M. Levy. "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proceedings of the IEEE/ACM International Symposium on Computer Architecture*, pp. 392-403, May 1995.

[11] Intel Ivy Bridge Specifications (two threads per core) http://ark.intel.com/

[12] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Multithreading," *IEEE Computer Architecture Letters*, 2013 (in press).

[13] L. O. Chua, "Memristor – The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[14] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209- 223, February 1976.

[15] T. Prodromakis, K. Michelakisy, and C. Toumazou. "Fabrication and Electrical Characteristics of Memristors with TiO_2/TiO_{2+x} Active Layers," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp.1520-1522, May 2010.

[16] Z. Biolek, D. Biolek, and V. Biolkova., "SPICE Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, Vol. 18, No .2, pp. 210-214, June 2009.

[17] G. M. Ribeiro *et al.*, "Designing Memristors: Physics, Materials Science and Engineering," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 2513–2516, May 2012.

[18] H.Y. Lee *et al.*, "Low Power and High Speed Bipolar Switching with a Thin Reactive Ti Buffer Layer in Robust HfO2 Based RRAM," *Proceedings of the IEEE International Electron Devices Meeting*, pp.1–4, December 2008.

[19] Y. F. Chang *et al.*, "Study of SiOx-Based Complementary Resistive Switching Memristor," *Proceedings of the Annual Device Research Conference*, pp. 49–50, June 2012.

[20] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211–221, January 2013.

[21] Y. N. Joglekar, and S. J. Wolf, "The Elusive Memristor: Properties of Basic Electrical Circuits," *European Journal of Physics*, Vol. 30, No. 4, pp. 661–675, July 2009.

[22] J. J. Yang *et al.*, "Engineering Nonlinearity into Memristors for Passive Crossbar Applications," *Applied Physics Letters*, Vol. 100, No. 11, pp. 113501–113501, March 2012.

[23] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 3, pp. 712–727, March 2006.

[24] FreePDK45 User Guide, April 2011, http://www.eda.ncsu.edu/wiki/FreePDK45.

[25] W. Zhao and Y. Cao, "New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration," *IEEE Transactions on Electron Devices*, Vol. 53, No. 11, pp. 2816–2823, January 2006.

[26] S. Kvatinsky, Y. H. Nacson, R. Patel, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Multithreading with Emerging Technologies – Dense Integration of Memory within Logic," (in submission).

[27] The gem5 Simulator System, May 2012, http://www.m5sim.org/.

[28] S. Li *et al.*, "McPAT: an Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 469-480, December 2009.

[29] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The Desired Memristor for Circuit Designers," *IEEE Circuits and Systems Magazine*, Vol. 13, No. 2, pp. 17-22, Second quarter 2013.

[30] M. K. Farrens and A. R. Pleszkun, "Strategies for Achieving Improved Processor Throughput," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 362-369, May 1991.

On the In-Die 3D Integration of Memory in CMOS Metal Layers and Its Implications on Processor Microarchitecture

Abstract

Over the recent years, new memory technologies such as RRAM and STT-MRAM have emerged. These non-volatile technologies have primarily been studied as replacement for flash, DRAM, and SRAM.

In this paper, we take a different approach and explore the tight integration of CMOS logic with emerging memory technologies inside the CMOS die. The physical properties of these emerging technologies allows them to be integrated into the metal layers of a CMOS die without requiring precious real-estate on the die's single transistor layer. As current CMOS technology offers about a dozen metal layers on top of a single transistor layer, such tight integration provides logic with extremely fast access to abundant memory resources. We refer to the tight integration of memory and logic as memory-intensive architecture.

We present the multi-state register as a building block for memory-intensive architectures stores multiple shadowed values for a single register in the metal layers of the die. Finally, we examine the implications of memory-intensive architecture on processor microarchitecture. We construct an in-order pipeline using the proposed multi-state register as pipeline registers and demonstrate how the use of multistate pipeline registers in this microarchitecture eliminates the need to flush the pipeline upon a thread switch in Switch-on-Event (SoE) multi-threading machines. We show that the resulting continuous flow multi-threading (CFMT) microarchitecture outperforms a traditional SoE design by 32% while consuming 8.5% less energy, thereby significantly increasing the performance to energy ratio.

Keywords Memristor, memristive device, STT-MRAM, PCM, RRAM, multithreading, CFMT, Memory Intensive Computing

1. Introduction

Emerging of nonvolatile memory technologies, such as RRAM [1-2] and STT-MRAM [3], offer high speed, high density, and low power memories, whose endurance is expected to reach that of SRAM and DRAM [4-5]. These technologies represent logical state as resistance on a conductor (variadic resistance for RRAM; switchable on/off modes for STT-MRAM) and, despite their use of different materials and physical properties, can be collectively refer to as memristive devices, or memristors[1] [6-9].

One interesting shared characteristic of these emerging memory technologies is that they are fabricated between two layers of metal. Existing CMOS-based dies are composed of a single layer of CMOS transistors with multiple metal layers stacked on top. The metal layers are etched to compose the wires that route the connections between the transistors on the CMOS layers, and modern processors may include up to a dozen metal layers [10]. Nevertheless, the metal layers are not fully utilized in many areas of the die.

In this paper, we propose in-die stacking of memory on top of logic by fabricating memristors in the metal layers of the die. The integrated *memory-intensive architecture* provides tight integration of logic with dense memory (memristors can be stacked on several layers to further increase density). This integration presents logic with fast access to abundant memory resources.

To make efficient use of memory-intensive architecture, we present the *multi-state register*. The multistate register extends a CMOS register with multiple shadow values stored in the memristor layers above. While logic can operate regularly on the active value stored in the CMOS layer, the active value can be switched with any of the shadow values within a single cycle. This building block thus provides a promptly switchable, multi-context state element. We have designed RRAM-based multistate registers using SPICE, including its physical layout, and present its performance, area, and energy characteristics.

Finally, we explore the implications of memory-intensive architecture on processor microarchitecture by replacing the pipeline registers in an in-order, switch-on-event (SoE), multi-threaded pipeline with multistate registers. The resulting continuous flow multi-threading (CFMT) pipeline can store multiple in-flight pipeline contexts. On a thread switch, the CFMT pipeline simply switches contexts in all multistate pipeline registers, thereby eliminating the need to flush the pipeline on every thread switch. We have implemented the CFMT pipeline using a modified gem5 simulator and show that it achieves high performance, while keeping the complexity and energy low as conventional SoE. Specifically, CFMT improves performance to energy ratio by an average of 44% (up to a 116% for floating point benchmarks) over an SoE pipeline.

[1] Although the use of the term memristor for these emerging memory elements is still under debate [9], we use it in this paper to refer to any device that is nonvolatile, dense, and resistance-based.

In summary, this paper makes the following contributions:

- Introduces the tight integration of logic with memristive memories by embedding memristors in the metal layer.
- Presents the multistate register as a building block for memory-intensive architecture.
- Explores the implications of memory-intensive architecture on processor microarchitecture through the evaluation of the continuous flow multithreading pipeline.

The rest of the paper is organized as follows: emerging memory technologies and the concept of memory intensive architecture are described in section 2. The multistate register is presented in section 3, following a case study of its integration in multithreaded processors in section 4. The performance and energy of CFMT are evaluated, respectively, in sections 5 and 6, following concluding remarks in section 7.

2. Stacking Memory Inside the Die

Emerging memory technologies are fabricated in the metal layers and can be integrated into a CMOS process, above the active silicon transistors, as shown in figure 1. In this section, we present how the integration into CMOS enables memory intensive architectures. This section then surveys the most promising emerging memory technologies and confronts their current status with the underlying assumptions regarding nonvolatility, relatively fast write and read (similar speed as an SRAM), practically unlimited write endurance, low energy, high density (stacked above the silicon transistors), good scalability, and compatibility with standard CMOS processes.

2.1 In-Die Integration

All emerging memory technologies are located on top of the silicon layer and can therefore be integrated with CMOS transistors right above them. This concept is illustrated in figure 1, where the emerging memory technologies (marked as 'memristors' in the figure) stacked in two layers as oxides sandwiched between two layers of metal. In this illustration, the memory devices are located between metal 3 and 4, and between metal 4 and 5. The memory devices can be located between any other metal layers as shown in section 3. The size of each of each memory device depends on the width of the metal wires, and is usually the minimal feature size of the technology. Emerging memory devices are therefore the smallest possible devices in each technology (usually the area of a single device is considered to be $4F^2$, where F is the feature size) allowing dense memories.

There are several memory cell structures. RRAM can be implemented as a crossbar structure, achieving a high density since the memory cell consists of only a single resistor. Using several stacked layers can further increase the density. Crossbars, however, suffer from the sneak path phenomenon [11], which increases power consumption and complicates the read process. To reduce the sneak path, nonlinearity is added by depositing additional materials above or below the memristive thin film, with no change in the density [12]. Alternatively, a transistor can be added to the memory cell as a selector, reducing the density of the memory array since transistors are usually larger in their area as compared to the emerging memory devices.

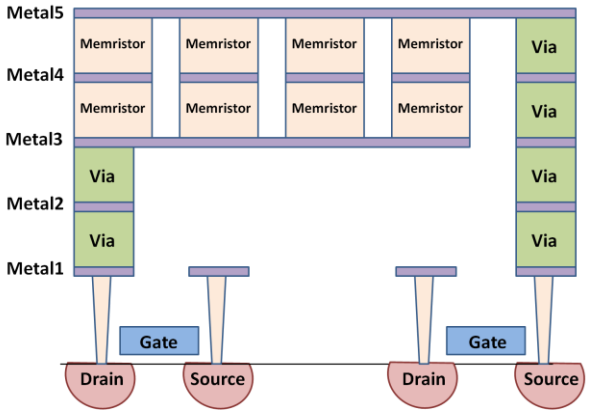


Figure 1: Physical structure of memory cells (memristors) on top of CMOS transistors. The memristors are located between metal layers 3 and 4, and 4 and 5.

2.2 Memory Intensive Architectures

A straightforward way is to use emerging memory technologies as improved replacements for existing memory technologies, and benefit from the improved characteristics, *e.g.*, higher density, no leakage, and high endurance. Using these technologies as SRAM replacements greatly increases on-die memory.

Standardization of emerging memory technologies has, however, implications for processor microarchitecture beyond conventional memory hierarchy. Emerging memory technologies can be used to enhance performance and decrease energy in memory intensive architectures [13], where processors are abundant with nonvolatile, fast memory, located on-top of the logic.

The additional memory increases the capacity of other elements within the processor, such as branch predictors, instruction queues, prefetching structures, reorder buffers, and other buffers. Additional memory elements can also be used to store data, which currently is not stored due to the limitations of conventional technologies. For example, the results of previously executed instructions can be stored for instruction reuse, to provide hardware memoization [14]. The states of different instructions for multiple threads can also be stored to enhance the performance of multithreaded processor as the case study presented in this paper.

2.3 Background on Emerging Memory Technologies

2.3.1 RRAM

Resistive Random Access Memory (ReRAM or RRAM) is based on dielectric materials - normally insulators - which can increase the conductance through a filament or conduction path, when a sufficiently high voltage is applied to the device for a sufficiently long time. This phenomenon is called resistive switching [1]. There are numerous types of materials that exhibit resistive switching, including binary transition metal oxides (*e.g.*, TiO_2, NiO), chalcogenides (*e.g.*, $Ge_2Sb_2Te_5$, AgInSbTe), and solid-state electrolytes (*e.g.*, GeS, GeSe). A physical schematic of a TiO_2 device is shown in Figure 2a.

Recently, Panasonic debuted a commercial RRAM product [15] for microcontrollers. The first commercial flash replacements are planned for 2015 [2]. Prototypes of RRAM demonstrate superiority over flash memories [16-17] and certain resistive switches exhibit comparable prop-

	Emerging					Commercialized			
	RRAM	PCM	STT-MRAM	CBRAM	FeRAM	SRAM	DRAM	Flash	HDD
Reciprocal Density [F^2]	< 4	4-16	20-60	6	6	140	6-12	4	2/3
Energy per bit [pJ]	0.1-3	2-25	0.1-2.5	2	2	0.0005	2	120	$1\text{-}10\cdot10^9$
Read time [ns]	0.1-10	10-50	10-35	50	50	0.1-0.3	10	$2.5\cdot10^4$	$5\text{-}8\cdot10^6$
Write time [ns]	0.1-10	50-500	10-90	50	75	0.1-0.3	10	10^5	$5\text{-}8\cdot10^6$
Retention	Years	Years	Years	Years	Years	As long as voltage applied	64 msec	Years	Years
Standby power	Zero	Zero	Zero	Zero	Zero	Cell leakage	Refresh power	Zero	~ 1W
Non-volatility	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Write endurance [cycles]	10^{12}	10^9	10^{15}	10^5	10^{15}	$>10^{16}$	$>10^{16}$	10^3-10^4	10^4
Present density	32 Gb	8 Gb	64 Mb	16 Gb	128 Mb	2-8 MB	8 GB	256 GB	>1TB
Multi-level	Yes	Yes	No	Yes	No	No	No	Yes	No

Table 1: Comparison of different memory technologies. The data is from [15-26], [46-50].

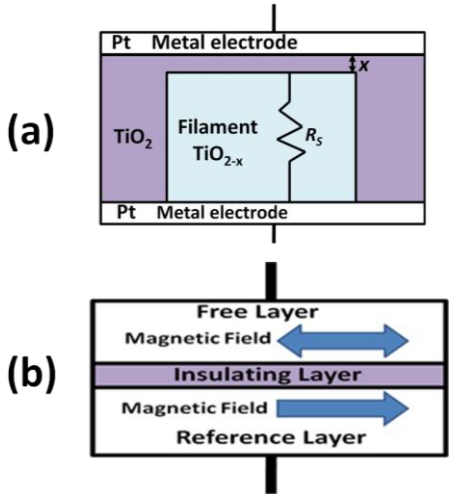


Figure 2: Different emerging memory devices. (a) A TiO_2 resistive RAM (RRAM) device and (b) Spin-Transfer Torque Magnetoresistive RAM (STT MRAM) cell.

erties to SRAM and DRAM [4, 18]. The properties of RRAM devices vary for different materials. Some materials are relatively fast. Tantalum oxide memristors, for example, can switch in approximately 100 picoseconds [19]. Although the endurance of RRAM is limited, it is considerably higher than flash (10^{12} as compared to 10^3 to 10^6 in flash), and will improve substantially in the near future, as required by caches and main memory [4]. The energy required to change the resistance of the device remains relatively high as compared to other memory technologies [18] but data retention does not consume static energy. As explained in section 2.1, the density of RRAM can be maximal for crossbars memories. Note that a single RRAM cell can store more than one bit, which further increases density.

2.3.2 STT-MRAM

Spin-Transfer Torque Magnetoresistive RAM (STT-MRAM) is a device usually consisting of two ferromagnetic metals with an oxide between the metal layers [3]. One metal has a fixed magnetic field and the other metal has a varying magnetic field, which changes according to the direction of the electric current flowing through it. When the two magnetic fields are parallel (in the same direction), the resistance of the device is relatively small. When the two magnetic fields are anti-parallel (in opposite directions), the resistance of the device is relatively high. Since there are only two stable states (parallel and anti-parallel), STT-MRAM behaves as a simple binary memory, and a single device cannot store more than a single bit. A physical schematic of an STT-MRAM cell is shown in figure 2b.

STT-MRAM is relatively fast (around 1 nsec write time) and has unlimited write endurance [5]. STT-MRAM suffers, however, from a low ratio between the high and low resistance, making it difficult to sense the data with process variations. The read and write energy of STT-MRAM is similar to the energy of RRAM. STT-MRAM cells require a transistor as a selector and their size is therefore characterized by the size of the CMOS transistor. Recently, Everspin Technologies, a spinoff of Freescale, announced the first commercial STT-MRAM memory. For this product, the read and write times are 35 nsec, with unlimited write endurance and capacity of 64 Mb [20].

2.3.3 Shared Properties of Practical Emerging Memory Technologies

There are several additional emerging memory technologies that share similar properties as the aforementioned technologies [21], such as Phase Change Memory (PCM) [22-24], Ferroelectric RAM (FeRAM or FRAM) [25] and Programmable Metallization Cell (PMC or CBRAM, sometimes considered as a specific type of RRAM) [26]. The properties of the emerging memory technologies described in this paper as well as conventional memory technologies are listed in Table 1.

Emerging memory technologies are still relatively slow as compared to SRAM. RRAM and STT-MRAM devices, however, have been demonstrated to exhibit relatively low write latencies, similar to SRAM. The write endurance of STT-MRAM and FeRAM is practically unlimited as desired for SRAM and DRAM replacement. Although the write endurance of RRAM is relatively high, it is still not sufficient to replace SRAM and DRAM. It is, however, expected that the write endurance reach that of SRAM [4].

Due to their nonvolatility, there is no leakage in emerging memory technologies. The dynamic energy per bit for emerging memory devices is, however, considerably higher than the energy per bit for SRAM. Since leakage power dominates in an SRAM cache, the total power dissipation for RRAM and STT-MRAM is lower than for SRAM. For example, STT-MRAM cache dissipates 60% less power than an SRAM cache with similar area [27].

RRAM and STT-MRAM are therefore candidates for SRAM replacements and are attractive technologies for use within a processor and fulfill the aforementioned requirements. Other technologies could also be used within a processor if speed, energy, endurance, and scaling issues are improved.

3. Memristor-Based Multistate Register

The introduction of massive amount of memory elements above the CMOS logic creates an opportunity to store data created at the CMOS level. The multistate register is a novel memory structure, used to store multiple bits within a single element. One set of bits is an active set, while the other sets are idle and stored for future use. The multistate register is a synchronous storage element, and the procedure to change the active set is therefore synchronized by a clock. The basic logical structure of a multistate register is shown in Figure 3.

An RRAM-based multistate register is shown in Figure 4 [28]. The multistate register primarily operates as a CMOS register. In this mode, the structure behaves as a standard D flip-flop, where a single bit is stored and is active while the idle states are stored within the RRAM crossbar array. Since the active state is stored within a CMOS register, the multistate register is compatible with any digital circuit. When global control circuitry triggers a change of the active set, the circuit stores the current bit of the register and reads out the value of the next active set from the internal RRAM-based storage. Switching between active bits therefore occurs in two phases, ideally taking one half-clock cycle per phase. While the delay of the read phase is limited by the CMOS register and is much lower than half a cycle, the write phase may require more than half a cycle, depending upon the switching time of the RRAM technology.

The RRAM-based multistate register utilizes a clocked CMOS register augmented by additional sense circuitry and global memristor select (MS) lines. The MS lines select individual RRAM devices within the crossbar memory co-located with the CMOS register. Local writes to the RRAM crossbar are controlled by the master stage within the CMOS register. The gates within the slave stage of the CMOS register are reconfigured to provide a built-in sense amplifier to read the RRAM crossbar array. The overhead of the additional circuitry is therefore relatively small, as shown in Figure 5a and 5bc for RRAM-based multistate registers with two physical design approaches – integrating RRAM devices between the first two metals or, alternatively, on the middle level metal layers. Using the lower metal layers approach is compatible with standard cell CMOS rules, but requires slightly more area than the middle level metal approach.

Although it is possible to design a CMOS SRAM-based multistate register (or any other conventional memory technology), emerging memory technologies enable high capacity multistate registers due to the high density and low leakage, as listed in Table 2 based on a 22 nm CMOS process. SRAM-based multistate register has large area, while the equivalent RRAM-based multistate register requires a relatively small area overhead. For example, a 64 state multistate register of SRAM that is based on a 22 nm CMOS process is 83 times larger than an RRAM-based multistate register.

A multistate register can be used for different purposes. In this paper, the application of multistate registers in the pipeline is described and demonstrated. In pipeline registers, the state of the instruction from the preceding pipeline stage is stored and transferred to the next pipeline stage. In a multistate pipeline register, additional instructions are also stored within the multistate register in background. The basic functionality of the pipeline is therefore unchanged. In the next section, we describe and evaluate

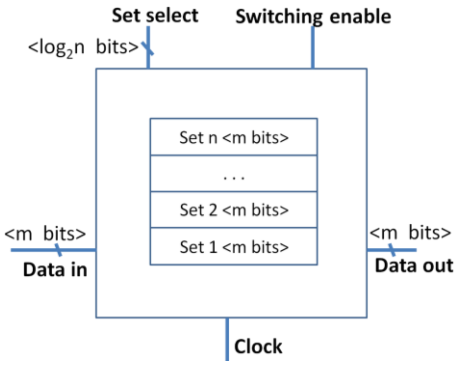


Figure 3: The logic structure of a multistate register. The size of each set is *m* bits, while *n* states are stored. The multistate register is synchronized by a clock and can switch the active set.

Register		Area [μm²]	Area Overhead
CMOS Register (1 state)		2.8	-
RRAM	4 states	3.9	39.3 %
	16 states	4.3	53.6%
	64 states	5.2	85.7%
SRAM	4 states	29.7	959.4%
	16 states	110.25	3837.6%
	64 states	432.61	15450.4%

Table 2: Area of a single bit Resistive RAM and SRAM for multistate registers based on a 22 nm CMOS process. The area of an RRAM-based multistate register is extracted by Cadence Virtuoso [28] and the area of an SRAM-based multistate register is extracted by NVSim [44].

Continuous Flow Multithreading (CFMT) [29], a novel microarchitecture that employs memristive multistate registers. In CFMT, a multistate register stores multiple machine states of different threads, where a single thread is active at a time, enabling higher throughput computing.

3.1 Multistate Register Evaluation

To evaluate the area, power, and performance of an RRAM-based multistate register, we used SPICE simulations along with cell layout. The RRAM device is modeled using the TEAM model [41]. The parameters of the resistive device are chosen to incorporate device nonlinearity into the current-voltage characteristic to reduce sneak paths. The multistate register is evaluated across a range of internal crossbar sizes (*i.e.*, different numbers of states per register). The transistor and cell track sizing information is from the FREEPDK45 Standard Cell Library [42] and scaled to a 22 nm technology. Circuit simulations utilize the 22 nm PTM CMOS transistor model [43].

The read operation of the multistate register requires 28.6 ps, equivalent to a 16 GHz clock frequency (the read operation is less than half a clock cycle). The latency of the write operation is assumed to vary from half a clock cycle to 4.5 clock cycles to fit different RRAM technologies, resulting different thread switch penalties that vary from a single clock cycle to five clock cycles.

To evaluate area, we assume that the area of a single memristor is 0.001934 μm² ($4F^2$, where F is the feature size) [18]. The area of an SRAM-based multistate register is extracted from NVSim [44] based on a 22 nm CMOS process. The use of SRAM-based multistate register in CFMT

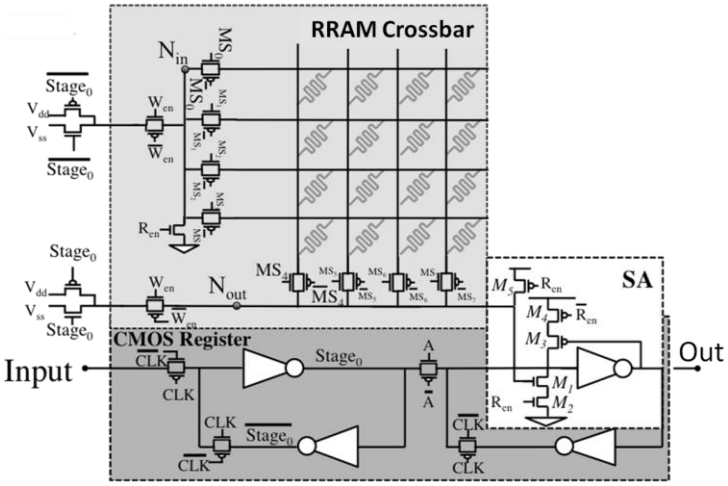


Figure 4: Complete circuit of a 16 state RRAM-based single bit multistate register consists of an RRAM-based crossbar array above a CMOS-based flip-flop, where the second stage (the slave) also behaves as a sense amplifier.

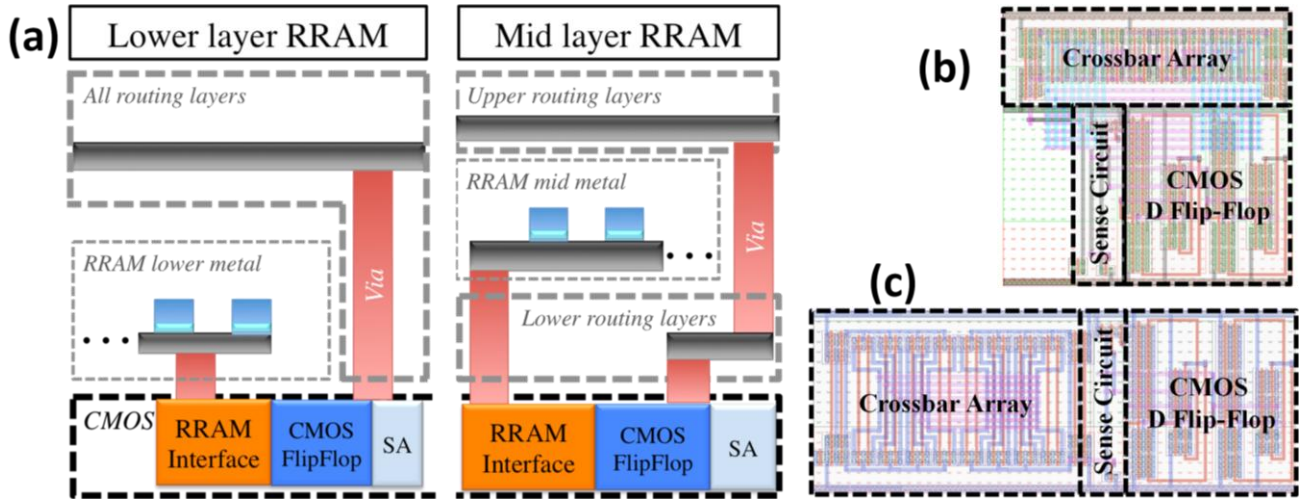


Figure 5: RRAM-based single bit multistate register. (a) Vertical layout of RRAM in multistate register, where the RRAM is within the lower metal layers (left) and middle metal layers (right), and physical layout of 64 state multistate register, where the RRAM is within (b) the lower metal layers (M1 and M2) and (c) middle metal levels (M2 and M3).

does not make sense due to its large area as listed in Table 2 and is not evaluated in this paper.

4. Test Case: Continuous Flow Multi-Threading (CFMT)

The availability of abundant state affects the design of processor microarchitectures. In this section we illustrate the potential benefit of memory intensive architecture through a microarchitecture case study. We demonstrate these benefits by replacing pipeline registers of an in-order Switch-on-Event multithreaded (SoE MT) processor with multistate registers. The resulting Continuous Flow Multithreading (CFMT) microarchitecture offers faster thread switch and higher performance, while saving power and maintaining the control as simple as SoE MT processors and. Unlike previous microarchitectures, which store the states of multiple threads (especially for GPUs) [30], CFMT stores intermediate (internal) states of each instruction within the pipeline.

4.1 Overview of Switch-on-Event Multithreading

Switch-on-Event Multithreading (SoE MT) is a multi-threading technique in which a processor hides the latency of long multi-cycle events (MCE), *e.g.*, an L1 cache miss, by executing instructions from different threads. When a sufficiently long MCE occurs, a thread switch is triggered and the MCE is executed in background. Otherwise (*i.e.*, when short latency instructions are executed), instructions from the same thread are fetched. During a thread switch, all of the sequential instructions from the thread that was switched are flushed and instructions from a different thread are fetched. The penalty of a thread switch is the time required to refill the pipeline.

A longer thread switch penalty reduces the performance. Hence, the number of pipeline stages is limited and, as a result, the frequency of the processor is restricted. Addi-

tionally, the effectiveness of a thread switch depends upon the ratio of the MCE latency and the switch penalty. For an MCE with latencies smaller than the switch penalty, it is preferable to stall the pipeline rather than switch to a new thread. For example, assume that MCEs are identified at the eighth pipeline stage (*i.e.*, seven clock cycles are required to refill the pipeline), the L1 cache miss penalty is 20 cycles, and the latency of an integer multiplication is three cycles, without pipelining the operation. For a cache miss MCE, a thread switch is worthwhile, but it is not worthwhile for an integer multiplication MCE. In this example, the latency of an L1 cache miss is hidden by the multithreading technique and does not influence the performance. Integer multiplication degrades the performance of the processor.

The performance of an SoE MT processor also depends upon the number of threads running within the processor. For a sufficient number of threads, instructions from other threads hide the latency of the MCE. In this case, the performance approximately saturates. For an insufficient number of threads, increasing the number of threads linearly improves performance.

Although an SoE MT is a simple technique that requires minimal control, it suffers from low performance and relatively high energy consumption due to repeated pipeline flushing. Other multithreading techniques, such as fine-grained multithreading [31] and simultaneous multithreading (SMT) [32], overcome the limitations of SoE MT. The complexity of these techniques, however, is greater. Furthermore, to achieve effective fine-grained multithreading, a large number of threads is required. An SMT processor, on the other hand, presents an opportunity to increase performance. Nevertheless, the complexity required to control a processor is significant, and the area and power consumption limit the processor to relatively few threads (*e.g.*, Intel's Ivy Bridge processor has only two threads per core [33]). SoE MT, fine-grained multithreading, and SMT are illustrated in Figure 6.

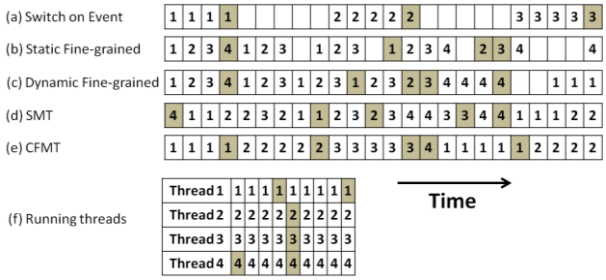


Figure 6: Timelines for different multithreading techniques with four threads (marked by the number). All processors run the same four threads as shown in (f). The latencies of the 'white' and 'shaded' instructions are, respectively, a single clock cycle and ten clock cycles. For an (a) SoE MT processor, the event that triggers a thread switch is the shaded instruction and the thread switch penalty is five clock cycles. (b) Static and (c) dynamic fine-grained multithreading achieves higher utility, while (d) SMT is optimized to achieve the maximum performance. (e) CFMT achieves high performance with simple switching and control mechanisms.

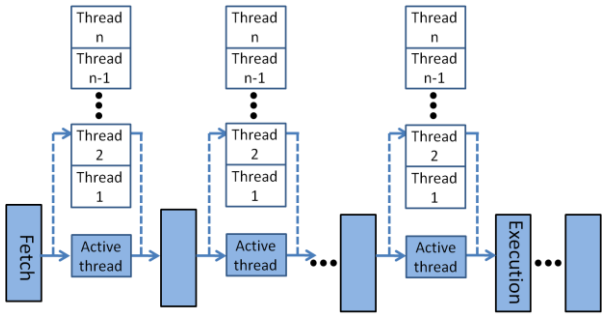


Figure 7: Continuous Flow Multithreading structure. A multistate pipeline register (MPR) is located between every two pipeline stages instead of a conventional pipeline register. The MPR stores the state of instructions from all supported threads within the machine when only a single thread is active at a time.

4.2 Introducing CFMT

Using multistate registers as pipeline registers in multithreaded processors can enhance performance by minimizing the switch penalty. In conventional pipelines, the pipeline registers are located between pipeline stages to store the state of the predecessor instructions before moving the state to the next pipeline stage. Conventional pipeline registers are replaced by multistate registers, as shown in Figure 7. The use of multistate registers instead of regular registers saves the state of the stalled threads in addition to the state of the active thread. The mechanism of thread switching is therefore different from a conventional SoE MT. Rather than flushing the pipeline, the states of the consecutive instructions are stored within the multistate registers in the memristive layer, locally near the relevant pipeline stage. On a thread switch rather than refilling the pipeline, instructions from the new active thread are read from the multistate registers, significantly reducing the thread switch penalty to the time required to read data from an multistate register. The conceptual behavior of CFMT is similar to the behavior of an SoE MT (see Figure 6a) with fewer bubbles, as shown in Figure 6e. Additionally, the novel switching mechanism helps conserve energy since reading and writing to the multistate registers consume less energy than refilling the entire pipeline and replaying the flushed instructions.

Lowering the thread switch penalty also allows new events to trigger a thread switch. With a conventional SoE MT, it is worthwhile to switch threads on events, when the latency is longer than the time required to refill the pipeline. With CFMT, the condition changes and it is effective to switch threads on events when the latency is longer than the time to switch an active thread within a multistate register. This condition allows having MCEs that would stall the pipeline in a conventional SoE MT. This improvement further increases the performance of the processor such as floating point operations and long latency integer operations. The analysis and evaluation of performance and power are presented in Sections 5 and 6.

In CFMT, the controller acts similar to a simple conventional SoE MT. The simplicity of CFMT is due to the use of an in-order pipeline, with only a single active thread at any given time. While the control mechanism is simple and the energy is low, CFMT offers substantially higher utilization and performance.

5. Methodology

5.1 Architecture

We have validated the required control mechanism and size of instruction state of a CFMT processor using an RTL implementation written in Verilog and simulated by ModelSim. To evaluate the performance of CFMT and compare it to SoE, the Gem5 simulator [35] has extended to support CFMT. The simulated processor has 13 pipeline stages and two levels of cache. All thread switching occurs in a unified execution and memory stage, located in the ninth pipeline stage. The parameters of the processor structure are listed in Table 3. The structure of the pipeline is similar to the ARM cortex-A8 processor [36] while the timing parameters of the execution units are extracted from the ARM cortex-A9 processor [37]. While ARM cortex-A8 is an in-order processor, the execution units of the out-of-order ARM cortex-A9 are faster and demonstrate the benefits from CFMT with shorter MCEs as well. The execution units are listed in Table 4. The hybrid branch predictor is chosen based on the ALPHA 21264 microprocessor [38] to achieve a high branch prediction rate. We simulate up to 32 threads per processor. Each workload is executed until the first thread completes 60 million instructions. We use ALPHA compiled SPEC CPU 2006 [39] benchmarks, generated by a Gem5 simulator. We use SPEC CPU 2006 for the performance evaluation since multi-programmed SPEC benchmarks have no resource sharing and are therefore more demanding for the cache and other resources.

In the CFMT processor, the pipeline registers and the register file are modeled as RRAM multistate registers. It is possible to also use memristive technologies for the caches, branch predictor, TLB, and other structures [40], but we consider these extensions beyond the scope of this paper. Nevertheless, these structures with memristive technologies will further reduce the energy without undue influence on performance.

5.2 Processor Implementation

Pipelining of the execution stage allows for more than a single instruction within the execution stage. Hence, a dependency check is performed prior to execution and independent sequential instructions enter the execution stage without a thread switch. When an instruction depends on a previous long latency instruction that is currently being executed, a thread switch is triggered. Exiting from the execution stage, however, is limited to a single instruction per clock cycle and is accomplished in-order. In the case of a conflict between instructions from different threads, the active thread is preferred. This does not starve the previous thread, since executed instructions from previous threads can exit the execution stage during thread switches. Additionally, instructions that trigger a thread switch and finish the execution continue to propagate within the pipeline after executing.

For CFMT, two switching policies are considered: an L1 cache miss as the only switching trigger and any instruction with latency longer than the thread switch penalty (including an L1 cache miss). Furthermore, different thread switch penalties have been evaluated for CFMT, varying from a single clock cycle to five clock cycles, to demonstrate different RRAM technologies. To maintain fairness, a thread switch is triggered every 500 cycles. In practice, this fairness mechanism is not required since thread switches are more frequent.

Pipeline depth	13 stages
Execution stage	9th stage
L1 cache	32kB, 4-ways, 2 cycles latency
L2 Cache	1MB, 8 ways, 20 cycles latency
Memory latency	200 cycles
Branch predictor	8kB gshare

Table 3: Parameters of the simulated processor.

Execution Unit	No. of Units	Latency [Cycles]	Pipelined?
Int ALU	1	1	NA
Int multiply	1	3	V
FP ALU	1	4	V
FP multiply	1	5	V
FP multiply double precision	1	6	V
FP div	4	15	X
FP div double precision	4	25	X
FP sqrt	4	17	X
FP sqrt double precision	4	32	X

Table 4: Execution units of the simulated processor. The latencies are taken from [37].

Pipeline stage energy	E_{stage}	15.7 pJ
# instructions per flush	k	7
# of bits per instruction state	m	300 bits
RRAM-based 16 threads MPR - write energy @ X clock cycles thread switch penalty	$E_{MPR,write}$	3 fJ @ 1 cycle 7.8 fJ @ 2 cycle 12.6 fJ @ 3 cycle 17 fJ @ 4 cycle 21.7 fJ @ 5 cycle
RRAM-based 16 threads MPR - read energy	$E_{MPR,read}$	1.3 fJ

Table 5: Energy evaluation. Extracted from McPAT and SPICE simulations for CMOS 22 nm process with a clock frequency of 3 GHz.

In CFMT, a branch misprediction is the only trigger for a pipeline flush. Although switching threads when identifying a branch instruction can eliminate the pipeline flush, it significantly increases the complexity of the CFMT control mechanism since more than a single thread is active within the pipeline. To maintain the simplicity of the control mechanism, we flush the pipeline on a branch misprediction and use a branch predictor.

The number of execution units (as listed in Table 4) is chosen to eliminate structural hazards due to the lack of an available execution unit for different threads. If the number of execution units is low, resource sharing limits the performance and the processor may stall until an execution unit becomes available.

Multistate registers for superscalar processors store more instructions for each thread (the number of instructions is the pipeline width). Although more instructions run through the pipeline and are stored within the multistate registers, the control mechanism remains the same and the complexity is therefore unchanged. The performance increases although the in-order execution mechanism has a greater number of dependencies.

5.3 Energy Methodology

To evaluate the energy of the processor for both SoE MT and CFMT, the McPAT modeling framework [45] is used for a CMOS 22 nm process with a clock frequency of 3 GHz. Since McPAT does not consider thread switching in its energy evaluation, the energy consumed by these operations is extracted from (2) and (3), as explained in section 6.2.1. The number of thread switches is evaluated by the performance simulator and the relevant parameters are extracted from SPICE simulations and McPAT. The

parameters used to evaluate the thread switch energy are listed in Table 5.

6. Evaluation

6.1 Performance

6.1.1 Analytic Evaluation

The performance of both SoE and CFMT depends upon the number of threads within the processor. Adding more threads, allows long latency instructions (MCE) to be hidden. Each additional thread increases the performance. In this case, the machine is *unsaturated*. Once the processor has a sufficient number of threads to completely hide the long latency instructions, the performance is almost constant and the pipeline is full at this point to capacity. Adding more threads to the machine does not increase performance. In this case, the machine is *saturated*. It is possible, however, that additional threads affect the cache behavior, increasing the cache miss rate, thereby lowering performance [34].

Our analytic analysis assumes a processor with different MCEs whose latencies are P_i. In that case, the performance (in cycles per instruction) for n running threads with similar periodic average behaviors is

$$CPI = \begin{cases} \dfrac{CPI_{ideal} + \sum_{i \in mce} r_i \cdot P_i + r_m \cdot MR(n) \cdot P_m}{n}, & unsaturated\,(n < N_{sat}) \\ CPI_{ideal} + \sum_{i \in unhidden,mce} r_i \cdot P_i + \sum_{i \in hidden,mce} r_i \cdot P_s, & saturated\,(n > N_{sat}) \end{cases} \quad (1)$$

where the parameters are those listed in Table 6. Note that the impact on CPI due to the memory instructions depends on both the memory access time and miss rate and therefore on the number of threads.

The performance of SoE and CFMT is shown in Figure 8, and is approximately the same when both processors operate in the unsaturated region. The CPI_{sat} of CFMT is lower than SoE for two reasons: more MCEs are considered as triggers for a thread switch, and the thread switch penalty is lower than in SoE.

6.1.2 Simulations

The performance of three selected benchmarks that demonstrate three different possible behaviors is shown in Figure 9. The instruction mix of *soplex.ref* (Figure 9a) is 15% floating point instructions, 29% memory instructions, and 56% integer instructions. In the saturation region, the L1 cache miss rate is approximately 11% to 24% and the influence of both floating point and memory events is relatively similar. Hence, the IPC is improved as more switching triggers are considered. The IPC in the saturation regions for SoE MT is 0.39, while the IPC of CFMT when only a cache miss triggers a thread switch, and CFMT with both cache miss and floating point triggers are, respectively, 0.45, and 0.59, showing a performance improvement of 47% and 15% for, respectively, CFMT with and without floating point triggers. Note that the maximum performance is limited by the thread switch penalty (a single clock cycle), instruction dependencies of a single cycle, and branch mispredictions.

The *libquantum* benchmark (Figure 9b) does not include floating point instructions (20% memory instructions and 80% integer instructions). The performance of both switching policies for CFMT is therefore identical. Due to the relatively high L1 miss rate of approximately 15% in the saturation region, CFMT improves the performance by 18% as compared to SoE MT.

CPI_{ideal}	Ideal CPI of the machine
n	Number of threads
$MR(n)$	Miss rate as a function of n
r_i	Frequency of mce instruction i
r_m	Frequency of memory accesses
P_i	Penalty of mce instruction i
P_m	Memory access penalty
P_s	Switch penalty
N_{sat}	Number of threads that distinct between the unsaturated and saturated regions

Table 6: Parameters for the analytic model of SoE MT and CFMT described by (1).

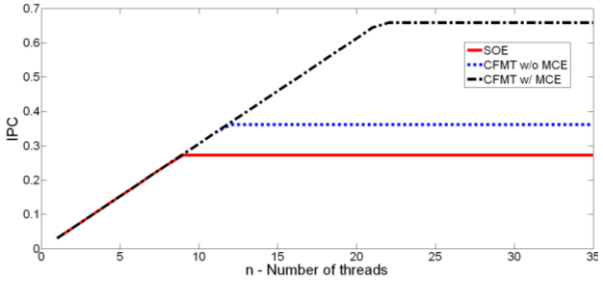


Figure 8: IPC (instruction per cycles) of Switch-on-Event Multithreading and Continuous Flow Multithreading as modeled in (1). The parameters are 30% memory instructions (r_m = 0.3) with an L1 cache miss rate of 50% and memory access time of 200 cycles (P_m = 200 cycles), r_1 = 20%, r_2 = 17%, P_1 = 3 cycles, and P_2 = 6 cycles.

For *gromacs* (Figure 9c) although memory instructions are frequent (42% of the total instructions), L1 cache misses are rare (an approximate L1 miss rate of 2% in the saturation region). Hence, the performance is influenced primarily by floating point operations (44%). CFMT with only L1 miss as a switching trigger performs similarly to SoE MT. CFMT with floating point as a switching trigger achieves a 55% performance improvement.

A comparison between the analytic model as presented in (1) to simulations is shown in Figure 10. The analytic model shows sufficient accuracy as compared to simulation results. The average difference between simulations and the analytic model is 2.6% for the entire IPC evaluation and 0.95% for the saturation region.

The speedup of the IPC in the saturation region as compared to SoE MT for numerous SPEC CPU 2006 benchmarks is shown in Figure 11 for an ideal multistate register (no thread switch penalty). The average performance speedup of CFMT (with MCE) as compared to SoE MT with RRAM multistate register (with thread switch penalty of a single clock cycle) is 32%. Floating point benchmarks, as marked in Figure 11, achieve an average performance improvement of 55% and 45% with, respectively, ideal and RRAM multistate registers. The maximum performance improvement is achieved for *zeusmp* (99% and 75% improvement, respectively, for ideal and RRAM multistate registers), where 33% of the instructions are floating point instructions, 25% are memory instructions, and the L1 cache miss rate in saturation is 17%.

The IPC for various thread switch penalties in CFMT is shown in Figure 12 for selected benchmarks. As expected, for CFMT the speedup decreases as the thread switch penalty increases [41]. The average speedup for various values of the thread switch penalty is listed in Table 7. Mixes of

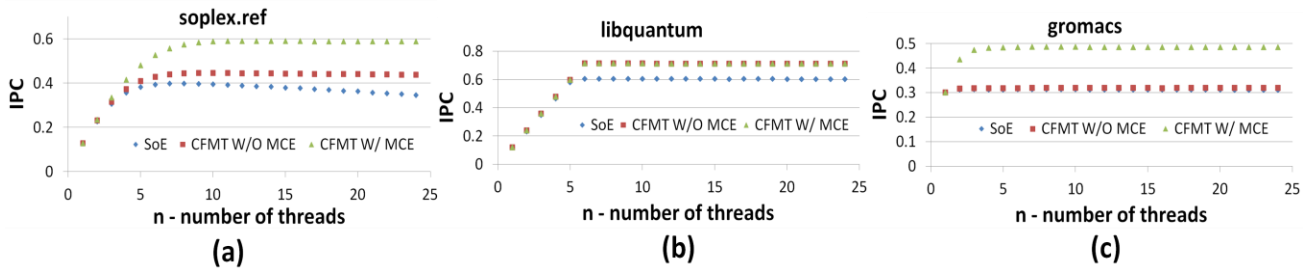


Figure 9: IPC vs. number of threads for different SPEC CPU 2006 benchmarks. (a) *soplex.ref*, where memory and floating point MCE both influence the performance, (b) *libquantum*, a benchmark of only integer and memory instructions (no other MCE), and (c) *gromacs*, a benchmark with a dominant floating point MCE over memory events. CFMT thread switch penalty is a single clock cycle. The other simulation parameters are as listed in Tables 4 and 5.

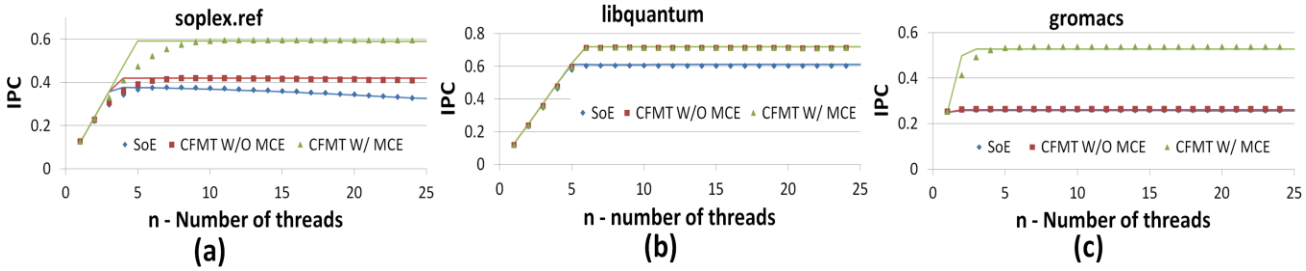


Figure 10: Comparison between the analytic model in (1) and simulation results for (a) *soplex.ref*, (b) *libquantum*, and (c) *gromacs*. The simulation results and analytic model are represented, respectively, by discrete dots and a straight line, exhibiting an average 4.2% difference.

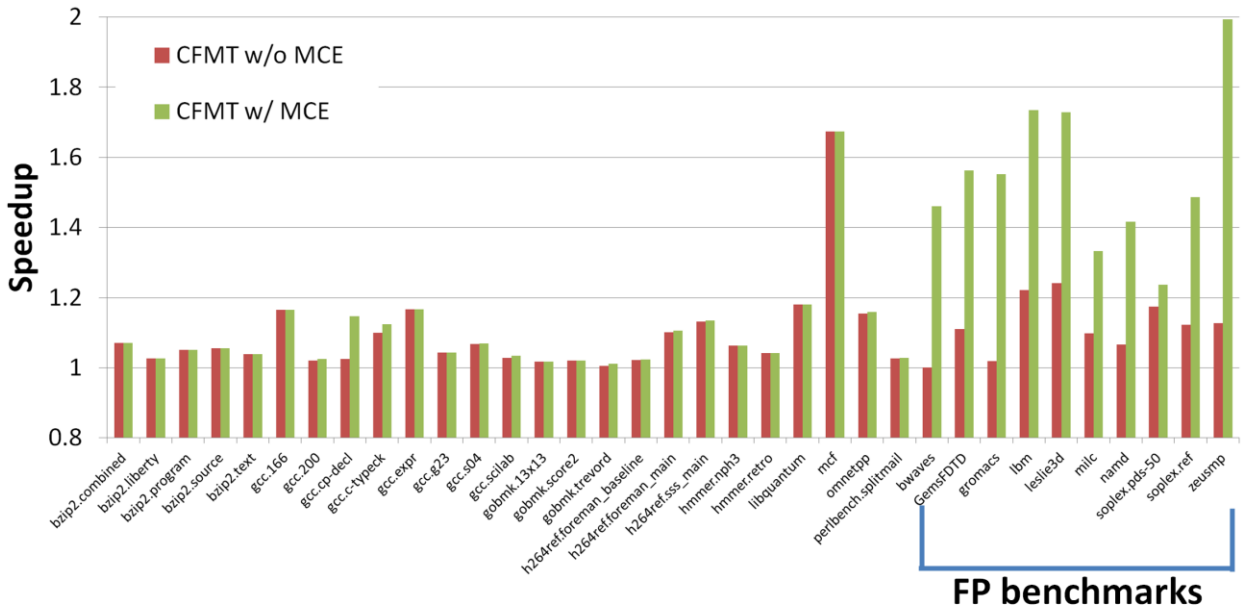


Figure 11: Speedup in the saturation region (number of threads is 16) for different SPEC CPU 2006 as compared to SoE MT with an ideal MPR (zero thread switch time).

different benchmarks have been tested, showing similar results. The IPC_{sat} is approximately an average of the results of the pure mix, as shown in Figure 13.

Increasing the cache size decreases the L1 cache miss rate, reducing the frequencies of the thread switches. The IPC for various L1 cache sizes, when only L1 cache miss is a switching trigger, is shown in Figure 14. Adding an L2 cache (the reference system in Figure 14) does not change the maximum performance, only the power consumption.

6.2 Energy

While CFMT significantly improves the performance as compared to SoE MT, it also dissipates less power. The total energy of the processor can be further decreased with emerging memory technologies. Since the control mechanism of both SoE MT and CFMT is similar, the main difference in energy is due to the thread switch mechanism. Additionally, the improved performance effectively lowers the leakage energy, further decreasing the energy.

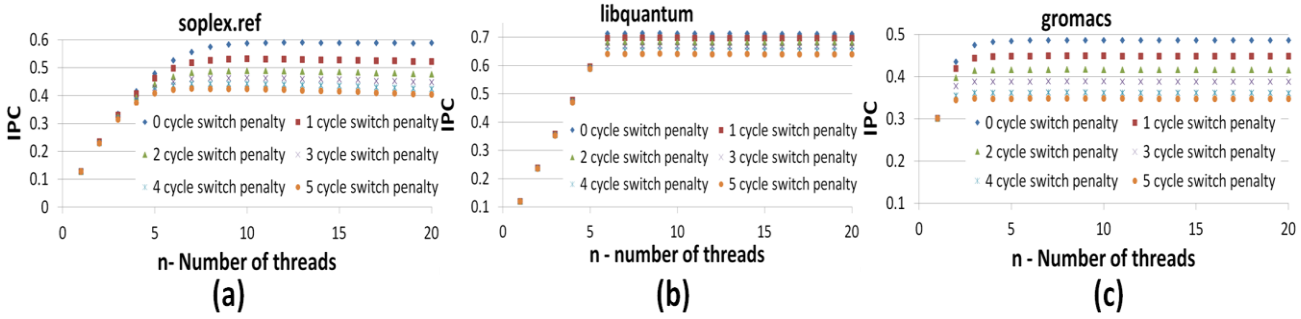


Figure 12: IPC vs. number of threads for various CFMT thread switch penalties. (a) *soplex.ref*, (b) *libquantum*, and (c) *gromacs*.

Benchmark	CFMT without MCE					CFMT with MCE				
Thread switch penalty [cycles]	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
libquantum	1.15	1.13	1.11	1.08	1.06	1.15	1.13	1.11	1.08	1.06
bwaves	1.01	1.01	1.01	1.01	1.00	1.35	1.24	1.07	1.04	1.01
milc	1.09	1.07	1.06	1.05	1.04	1.23	1.15	1.10	1.08	1.06
zeusmp	1.12	1.10	1.08	1.06	1.05	1.75	1.55	1.38	1.29	1.21
gromacs	1.02	1.02	1.01	1.01	1.01	1.43	1.33	1.24	1.16	1.11
leslie3d	1.17	1.15	1.12	1.10	1.07	1.51	1.38	1.27	1.20	1.14
namd	1.09	1.08	1.06	1.05	1.04	1.32	1.21	1.12	1.07	1.04
soplex.pds-50	1.17	1.14	1.12	1.09	1.07	1.22	1.18	1.13	1.11	1.08
lbm	1.19	1.16	1.13	1.10	1.07	1.54	1.30	1.21	1.15	1.10
bzip2.combined	1.09	1.07	1.06	1.05	1.04	1.09	1.07	1.06	1.05	1.04
gcc.166	1.14	1.12	1.10	1.08	1.06	1.14	1.12	1.10	1.08	1.06
gobmk.trevorc	1.15	1.12	1.10	1.08	1.06	1.15	1.13	1.10	1.08	1.06
h264ref.foreman_baseline	1.09	1.08	1.06	1.05	1.04	1.10	1.08	1.07	1.05	1.04
GemsFDTD	1.30	1.25	1.20	1.15	1.11	1.68	1.49	1.30	1.17	1.10
hmmer.nph3	1.17	1.15	1.12	1.09	1.07	1.17	1.15	1.12	1.09	1.07
soplex.ref	1.15	1.12	1.10	1.08	1.06	1.40	1.28	1.21	1.15	1.10
gcc.c-typeck	1.13	1.11	1.09	1.07	1.05	1.15	1.13	1.11	1.09	1.07
Average	1.13	1.11	1.09	1.07	1.05	1.32	1.23	1.16	1.11	1.08

Table 7: Performance speedup for various CFMT thread switch penalties.

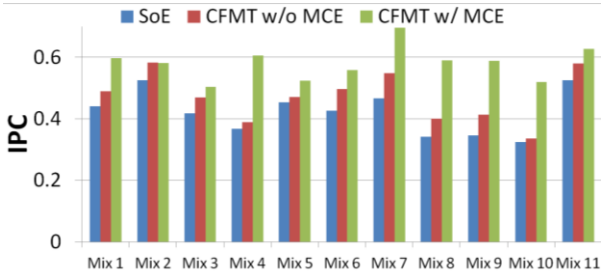


Figure 13: IPC in saturation for different benchmark mixes (16 threads, four each) of SPEC CPU 2006, as listed in Table 8. The speedup is approximately the average speedup of the different benchmarks in the mix.

Mix No.	Tested Benchmarks (16 Threads)
1	4 milc, 4 soplex.ref, 4 namd, 4 gcc.g23
2	4 libquantum, 4 gcc.166, 4 bzip2.program, 4 gobmk.nngs
3	8 lbm, 8 hmmer.nph3
4	4 lbm, 4 zeusmp, 8 libquantum
5	4 namd, 4 zeusmp, 4 GemsFDTD, 4 leslie3d
6	8 lbm, 4 GemsFDTD, 4 leslie3d
7	8 gromacs, 8 namd
8	8 libquantum, 4 namd, 4 bzip2.program
9	4 libquantum, 4 gcc.166, 4 bzip2.program, 4 gobmk.nngs
10	4 gcc.s04, 4 gobmk13x13, 4 sjeng, 4 zeusmp
11	4 hmmer.nph3, 4 libquantum, 4 lbm, 4 gromacs

Table 8: Different SPEC CPU 2006 mixes.

6.2.1 Analytic Evaluation

The energy of the thread switch in SoE MT is primarily due to the instruction flush. The flushed instructions are replayed when the thread is active again, going through the same pipeline stages an additional time. The average thread switch energy for SoE MT is therefore

$$\Delta E_{flush} = \frac{k \cdot (k+1)}{2} \cdot \overline{E_{stage}}, \qquad (2)$$

where $\overline{E}_{stage}$ is the average energy consumed in a single pipeline stage within a single clock cycle, and k is the number of flushed instructions during each thread switch. The energy of a thread switch in CFMT is the energy required to both read the state of the new active thread and write the state of the previous active thread. Formally, the average thread switch energy for CFMT is

$$\Delta E_{CFMT} = m \cdot \left(E_{MPR,write} + E_{MPR,read} \right), \qquad (3)$$

where m is the average number of bits required to represent the state of an instruction within a pipeline, $E_{MPR,write}$ and $E_{MPRread}$ are, respectively, the energy of a write and read in a single-bit multistate pipeline register. For the RRAM-based multistate register shown in Figure 4, the energy depends on the CMOS flip flop energy, and the resistance of the resistive switches and the switching energy of the memory devices, which determine, respectively, the read and write energy. Furthermore, the improved IPC of CFMT reduces the run time of the processor workload and the static energy due to leakage current.

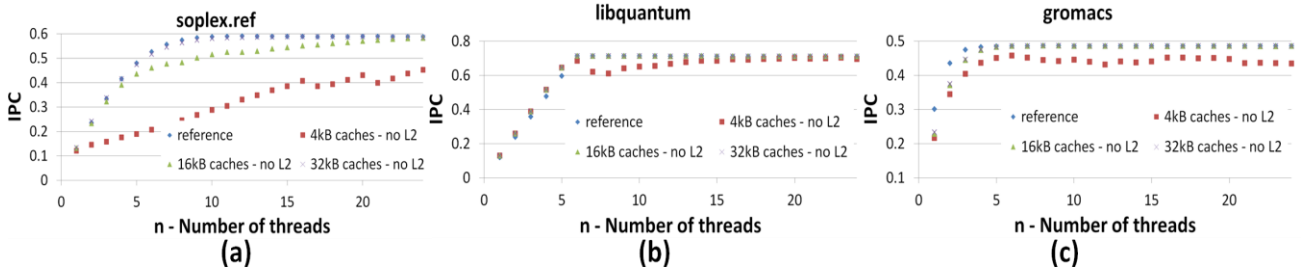


Figure 14: IPC vs. number of threads for various L1 cache sizes (4 kB, 16 kB, and 32 kB) for CFMT without MCE and without L2 cache, as compared to a reference machine with L2 cache as listed in Table 4. (a) *soplex.ref*, (b) *libquantum*, and (c) *gromacs*.

Benchmark	SoE MT [pJ/inst.]	CFMT				
		RRAM MPR – various thread switch latencies				
		1 cycle [pJ/inst.]	2 cycles [pJ/inst.]	3 cycles [pJ/inst.]	4 cycles [pJ/inst.]	5 cycles [pJ/inst.]
Libquantum	15.922	15.26	15.352	15.441	15.5315	15.621
bwaves	21.641	20	20.402	21.178	21.3581	21.568
milc	26.367	24.72	25.1	25.395	25.6783	25.744
zeusmp	22.651	19.04	19.642	20.312	20.7422	21.154
gromacs	32.871	30.07	30.558	31.052	31.5899	31.904
leslie3d	29.54	26.67	27.203	27.778	28.1692	28.547
namd	24.945	23.15	23.634	24.1	24.4246	24.608
soplex.pds-50	19.639	18.47	18.66	18.872	19.0249	19.166
lbm	24.308	21.34	22.306	22.832	23.1923	23.467
bzip2.combined	24.583	24.08	24.143	24.209	24.2771	24.344
gcc.166	20.064	19.38	19.477	19.568	19.6613	19.753
gobmk.trevorc	26.445	25.44	25.579	25.712	25.8522	25.984
h264ref.foreman_baseline	27.991	27.16	27.272	27.439	27.5789	27.645
GemsFDTD	28.574	23.35	24.332	25.715	27.3383	27.904
hmmer.nph3	28.353	26.02	26.153	26.284	26.4131	26.545
soplex.ref	23.78	21.51	21.782	22.137	22.4232	22.74
gcc.c-typeck	21.783	20.94	21.038	21.139	21.2413	21.347
Average	24.674	22.74	23.096	23.48	23.7939	24.002

Table 9: Energy per instruction for various SPEC CPU 2006 benchmarks in the saturation region.

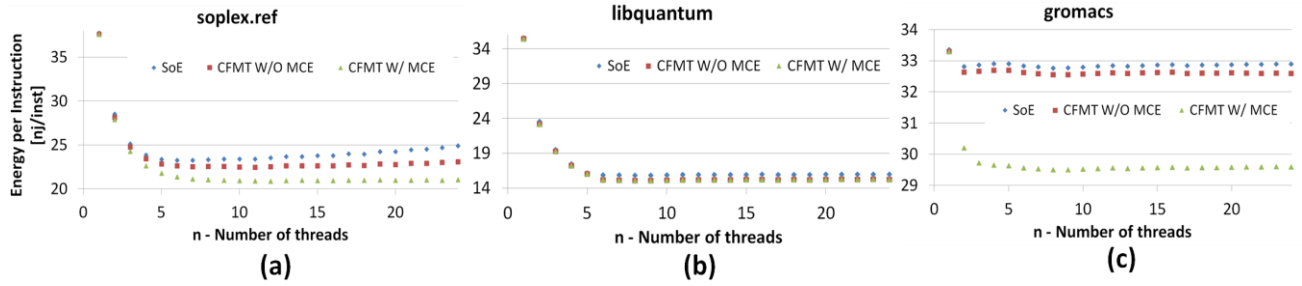


Figure 15: Energy per instruction vs. number of threads for SoE MT and CFMT for selected benchmarks. (a) *soplex.ref*, (b) *libquantum*, and (c) *gromacs*.

6.2.2 Experimental Results

The energy per instruction for a varying number of threads is shown in Figure 15, exhibiting a lower energy per instruction when more threads are running within the machine. The measured energy per instruction for various SPEC CPU 2006 benchmarks is listed in Table 9. The energy per instruction is reduced on average by 8.5% for CFMT with an RRAM-based multistate register as compared to SoE MT and up to 19% for *zeusmp*. The energy reduction is primarily due to the lower static energy.

7. Conclusions

The in-die integration of emerging memory technologies with CMOS paves the way for memory intensive architectures. Memory intensive architectures use novel memory elements to store data not stored in conventional architectures to enhance performance, while reducing energy. Rather than using additional memory solely to increase the capacity of the traditional memory hierarchy, the additional memory is used for novel architectural opportunities.

As an example of memory intensive computing, the combination of a novel memory structure, multistate pipeline register (MPR), with a novel microarchitecture, Continuous Flow Multithreading (CFMT), exhibits a 32% performance improvement with a reduction in energy. The performance per energy support the use of CFMT in low power machines.

CFMT is a single example of a memory intensive architecture. Numerous other applications of multistate register and other memory elements based on emerging memory technologies are possible. These novel architectures will improve both performance and energy and extend CMOS by adding to it complementary technology.

References

[1] R. Waser and M. Aono, "Nanoionics-based Resistive Switching Memories," *Nature Materials*, Vol. 6, pp. 833-840, November 2007.

[2] K. Tsutsui, "Focus on Strengths and Weaknesses of ReRAM," *Proceedings of the Flash Memory Summit*, August 2013.

[3] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-Transfer Torque Switching in Magnetic Tunnel Junctions and Spin-Transfer Torque Random Access Memory," *Journal Of Physics: Condensed Matter*, Vol. 19, No. 16, pp. 1-13, 165209, April 2007.

[4] J. Nickel, "Memristor Materials Engineering: From Flash Replacement Towards a Universal Memory," *Proceedings of the IEEE International Electron Devices Meeting*, December 2011.

[5] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, "Spin Transfer Torque (STT) MRAM-based Runtime Reconfiguration FPGA Circuit," *ACM Transactions on Embedded Computing Systems*, Vol. 9, No. 2, pp. 14:1-14:16, October 2009.

[6] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The Desired Memristor for Circuit Designers," *IEEE Circuits and Systems Magazine*, Vol. 13, No. 2, pp. 17-22, second quarter 2013.

[7] L. O. Chua, "Memristor – The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[8] L. O. Chua and S. M. Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209- 223, February 1976.

[9] L. O. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A: Materials Science & Processing*, Vol. 102, No. 4, pp. 765-783, March 2011.

[10] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappel, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunthar, T. Piazza, and T. Burton, "Haswell: The Fourthe-Generation Intel Core Processor," *IEEE Micro Magazine*, Vol. 34, No. 2, pp. 6-20, March/April 2014.

[11] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-Path Constraints in Memristor Crossbar Arrays," *Proceedings of the IEEE International Symposium on Information Theory*, pp. 156-160, July 2013.

[12] J. J. Yang, M.-X. Zhang, M. D. Pickett, F. Miao, J. P. Strachan, W.-D. Li, W. Yi, D. A. A. Ohlberg, B. J. Choi, W. Wu, J. H. Nickel, G. Medeiros-Riberio, and R. S. Williams, "Engineering Nonlinearity into Memristors for Passive Crossbar Applications," *Applied Physics Letters*, Vol. 100, No. 11, pp. 113501–113501, March 2012.

[13] U. Weiser, "Memory Intensive Computing," *ISCA Keynote*, June 2013.

[14] E. Schnarr and J. R. Larus, "Fast Out-Of-Order Processor Simulation Using Memoization," *Proceedings of the International Conference on Architectural Support of Programming Languages and Operationg Systems*, pp. 283-294, December 1998.

[15] http://panasonic.co.jp/

[16] "Elpida Memory Develops Resistance RAM Prototype," press release: http://www.elpida.com/en/news/2012/01-24r.html

[17] T-Y. Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. K. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C.-Y. Chen; M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau; X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, K, "A 130.7 mm^2 2-Layer 32 Gb ReRAM Memory Device in 24 nm Technology," *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 1-14, February 2013.

[18] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive Devices for Computing," *Nature Nanotechnology*, Vol. 8, pp. 13-24, January 2013.

[19] C. Torrezan, J. P. Strachan, G. Medeiros-Riveiro, and R. S. Williams, "Sub-nanosecond Switching of a Tantalum Oxide Memristor," *Nanotechnology*, Vol. 22, No. 48, pp. 1-7, December 2011.

[20] http://www.everspin.com

[21] M. H. Kryder and C. S. Kim, "After Hard Drives—What Comes Next?," *IEEE Transactions on Magnetics*, Vol. 45, No. 10, pp. 3406-3413, October 2009.

[22] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 2-13, June 2009.

[23] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 24-33, June 2009.

[24] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M.-G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," *Proceedings of the IEEE International Solid-State Circuits Conference*, pp.46-48, February 2012.

[25] J. F. Scott and C. A. Paz de Araujo, "Ferroelectric Memories," *Science*, Vol. 246, No. 4936, pp. 1400-1405, December 1989.

[26] M. N. Kozicki and W. C. West, "Programmable Metallization Cell Structure and Method of Making Same," *U. S. Patent No. 5,761,115*, June 1998.

[27] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs," *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, pp. 239-249, February 2009.

[28] R. Patel, E. G. Friedman, A. Kolodny, and S. Kvatinsky, "Multistate Register Based on Resistive RAM (ReRAM) – A Novel Digital Circuit," *IEEE Transactions on Very Large Scale Integration (VLSI)*, (in press).

[29] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Multithreading," *IEEE Computer Architecture Letters*, 2013 (in press).

[30] W.-K. S. Yu, R. Huang, S. Q. Xu, S.-E. Wang, E. Kan, and G. E. Suh, "SRAM-DRAM Hybrid Memory with Applications to Efficient Register Files in Fine-Grained Multi-Threading," *Proceedings of the Annual International Symposium on Computer architecture*, pp. 247-258, June 2011.

[31] T. Ungerer, B. Robic, and J. Silic, "A Survey of Processors with Explicit Multithreading," *ACM Computing Surveys*, Vol. 35, No. 1, March 2003.

[32] D. M. Tullsen, S. J. Eggers, and H. M. Levy, " Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 392-403, May 1995.

[33] Intel's Ivy Bridge Specifications: http://ark.intel.com/

[34] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser, "Many-Core vs. Many-Thread Machines: Stay Away From the Valley," *Computer Architecture Letters*, Vol. 8, No. 1, pp. 25-28, May 2009.

[35] http://www.m5sim.org/

[36] "Cortex™-A8 Specification Summary", http://www.arm.com, 2011.

[37] "Cortex™-A9 Floating-Point Unit, Technical Reference Manual", http://www.arm.com, 2012.

[38] R. E. Kessler, E. J. McLellan, and D. A. Webb.,"The Alpha 21264 Microprocessor Architecture," *Proceedings of the IEEE International Conference on Computer Design*, pp. 90-95, October 1998.

[39] SPEC CPU2006 benchmark suite. http://www.spec.org/cpu2006/

[40] X. Guo, E. Ipek, and T. Soyata, “Resistive Computation: Avoiding the Power Wall with Low-Leakage STT-MRAM Based Computing,” *Proceedings of the International Symposium on Computer Architecture*, pp. 371-382, June 2010.

[41] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.

[42] FreePDK45 User Guide, April 2011, http://www.eda.ncsu.edu/wiki/FreePDK45.

[43] W. Zhao and Y. Cao, “New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration,” *IEEE Transactions on Electron Devices*, Vol. 53, No. 11, pp. 2816–2823, January 2006.

[44] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31, No. 7, pp. 994-1007, July 2012.

[45] S. Li, J.-H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore architectures," *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, pp.469-480, December 2009.

[46] T*he International Technology Roadmap for Semiconductor (ITRS): 2011 Edition*, Semiconductor Industry Association, 2011.

[47] "Mobile 3rd Generation Intel® Core™ Processor Family Datasheet," Intel, April 2012.

[48] http://www.samsung.com

[49] http://www.fujitsu.com

[50] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi. “Hybrid Checkpointing using Emerging Non-Volatile Memories for Future Exascale Systems,” *ACM Transactions on Architecture and Code Optimization (TACO)*, Vol.8, No. 2, Article 5, 29 pages, July 2011.

Chapter 4 Conclusions and Future Work

Memristors add new capabilities to CMOS technology and are projected to be commercially available in the near future. In this thesis, we investigate these new properties and the implications of the new capabilities of VLSI systems in the memristor era. This combination of nonvolatile, dense, fast, and low power devices that are used both in storage systems and embedded on-top of CMOS, are a disruptive technology, changing the way computers are organized today.

In this thesis, different families to perform computation with memristors, both for integration of memristors with CMOS and for logic within memory are presented. The integration between CMOS and memristors is beneficial to increase the logic gate count of the same area (increase logic density), even without shrinking CMOS transistors. This approach is beneficial to extend Moore's law when CMOS scaling becomes problematic or to go beyond Moore's law and increase the number of logic gates by more than the traditionally factor of two.

A different approach, investigated in this research, is using memristive-only logic for logic within the memory. This approach enables non-von Neumann architectures of in-memory computing (also named process in-memory). Unlike previously proposed in-memory computing architectures, using memristive memory does not require additional circuitry or changing the memory cell or the structure of the memory array. Logic within the memory based on memristors is therefore a pure in-memory processing and not an integration of computing engines and memory cells. It is still required to define the complete architecture for memristive computing in-memory, including its instruction set, control, and investigate the appropriate applications for it to be beneficial in terms of performance and energy. This research is out of the scope of this thesis and discussed as a future work in Chapter 4.2.

Memristors can be used as enablers to other non-von Neumann architectures. For example, memristors can be used in neuromorphic systems (hardware systems that try to mimic the brain) and hardware neural networks. Memristors are primarily used to implement synapses in these systems; it is also possible to use memristors as part of the neuron circuit as well. Although the use of memristors as synapses is out of the

scope of this dissertation, we investigate these architectures as well, as described in Chapter 4.1.

The use of memristors as enablers to novel architectures that integrate together CMOS transistors and memristor technologies is named "Memory Intensive Computing" and includes numerous possible architectures and microarchitectures. In this dissertation, we present a novel memory structure, the multistate register, and use it to enhance the performance of multithreaded processors (CFMT). CFMT is only a single example of a memory intensive architecture. There are many other possible applications for multistate registers, as further described in Chapter 4.2.

We believe that the proposed applications for memory intensive computing in this research are only the tip of the iceberg, and in the near future many other exciting novel applications will be proposed, changing the structure and architecture of computers and VLSI systems.

4.1 Research that is not Part of This Thesis

As part of this research about memristors and their applications, other aspects that are not part of this dissertation are also investigated. One aspect we explore is memristive crossbar memories from information theory perspective. We analyze the read and write operations in such memory arrays, defining the limitations of read and write operations due to sneak paths, and investigating how the data stored within the memristors influence the read and write operations. In [51] and [52], we investigate the read operation of memristive crossbar memories.

Another aspect of memristive applications is neuromorphic systems. In [18], we propose a novel synapse circuit, consists of a single memristor and two CMOS transistors. The proposed synapse is suitable for gradient descent learning and can therefore be used for the execution of numerous machine learning algorithms, *e.g.*, back propagation.

4.2 Future Research Ideas

Memory Intensive Architectures

There are many possible memory intensive architectures that need to be investigated, designed, and evaluated. For example, multistate registers can be used for additional microarchitectures (in addition to CFMT that is shown in this thesis). Using multistate registers as pipeline registers (MPR) is beneficial not only for SoE MT, but also for simultaneous multithreading (SMT). MPR-based SMT will have lower control complexity and the energy will therefore be lower, while the performance remains the same. Multistate registers can be used for other applications as well, such as register files, branch predictors, and transactional memories.

There are also other possible memory intensive architectures, including both von Neumann machines and other machines. It is possible to use memristive memories for data flow processors, associative processors, and for reconfigurable machines.

In-Memory Computing with Memristors

In this thesis, three different logic families for logic within a memristive memory are described. Different Boolean functions are executed with these logic families. The control of these logic families, however, is not discussed. It is required to develop a complete architecture for in-memory computing, using the basic memristive logic gates. Algorithms to execute any Boolean operation need to be develop, new instruction set is required, a compiler needs to be designed, and the system structure including the interface between the CPU to the memristive memory needs to be defined.

Another relevant research direction is investigating the application space. The idea is to investigate the dependency of performance of different applications in memory accesses and the structure of the memory system. The target of this research direction is to identify and classify applications where in-memory computing is beneficial both in terms of energy and performance.

Design of Multistate Registers in Different Technologies

In this thesis, a multistate register with RRAM crossbar on top of a CMOS D flip flop is presented. The implementation of multistate registers is, however, not limited

to this structure. Multistate registers can be used in different technologies, memristive (*e.g.*, STT MRAM, unipolar RRAM, PCM, *etc.*) and non-memristive (*e.g.*, SRAM). Each design has its own pros and cons (*e.g.*, area, power, and speed).

Memristor Modeling

While the TEAM model is widely used and has its advantages, it is limited to current-controlled bipolar memristors. Other models are also required to model the behavior of other memristive devices (*e.g.*, voltage-controlled, unipolar, and binary). Additionally, numerous models have been proposed since the TEAM model was published and it is worthwhile to compare them to the TEAM (a partial comparison of the TEAM model to other models has been done by Ascoli *et al.* in [53]).

Memristor Memories

The design of memristive memories is usually investigated for flash or DRAM replacements. There are, however, many other possibilities to design different memory structure (the multistate register is an example to this approach). Additionally, it is possible to design different structures for memory systems. For example, designing a three dimensional structure of the cache organization.

Another relevant topic is developing design tools for memory designers. These tools are used to evaluate the properties of the memristive memory, including its unique behavior (*i.e.*, sneak paths). There are different types of required tools, from high level tools that roughly evaluate the power and area of the entire memory, to low level simulators that considers the actual behavior of each memory cell.

References

[1] S. E. Thompson and S. Parthasarathy, “Moore's Law: the Future of Si Microelectronics,” *Materials Today*, Vol. 9, No. 6, pp. 20-25, June 2006.

[2] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, Vol. 23, No. 1, pp. 20-24, March 1995.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 365-376, June 2011.

[4] V. F. Pavlidis and E. G. Friedman, *Three-Dimensional Integrated Circuit Design*, Morgan Kaufmann, 2009.

[5] J. Scaramuzzo, "The Flash Transformed Data Center," *Proceedings of the Nonvolatile Workshop*, March 2014.

[6] J. Akerman, "Toward a Universal Memory," *Science*, Vol. 308, No. 5721, pp. 508-510, April 2005.

[7] L. O. Chua, “Memristor – the Missing Circuit Element,” *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507-519, September 1971.

[8] L. O. Chua and S. M. Kang, “Memristive Devices and Systems,” *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209-223, February 1976.

[9] D. B. Strukov, G. S.Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found,” *Nature*, Vol. 453, pp. 80-83, May 2008.

[10] D. Sacchetto, M. H. Ben-Jamaa, S. Carrara, G. DeMicheli, and Y. Leblebici, "Memristive Devices Fabricated with Silicon Nanowire Schottky Barrier Transistors," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 9-12, May/June 2010.

[11] K. A. Campbell, A. Oblea, and A. Timilsina, "Compact Method for Modeling and Simulation of Memristor Devices: Ion Conductor Chalcogenide-based Memristor Devices," *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 1-4, June 2010.

[12] X. Wang, Y. Chen, H. Xi, and D. Dimitrov, “Spintronic Memristor through Spin-Torque-Induced Magnetization Motion,” *IEEE Electron Device Letters*, Vol. 30, No. 3, pp. 294-297, March 2009.

[13] R. Waser, "Resistive Non-Volatile Memory Devices," *Microelectronic Engineering*, Vol. 86, No. 7-9, pp. 1925-1928, July-September 2009.

[14] L. O. Chua, "Resistance Switching Memories are Memristors," *Applied Physics A: Materials Science & Processing*, Vol. 102, No. 4, pp. 765-783, March 2011.

[15] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP Variations with Memristors for Spiking Neuromorphic Learning Systems," *Frontiers in Neuroscience*, Vol. 7, No. 2, pp. 1-15, February 2013.

[16] A. Afifi, A. Ayatollahi, and F. Raissi, "Implementation of Biologically Plausible Spiking Neural Network Models on the Memristor Crossbar-Based

CMOS/Nano Circuits," *Proceedings of the European Conference on Circuit Theory and Design*, pp. 563-566, August 2009.

[17] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Letters*, Vol. 10, No. 4, pp. 1297-1301, March 2010.

[18] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-Based Multilayer Neural Networks with Online Gradient Descent Training," *IEEE Transactions on Neural Networks* (in review).

[19] H.-K. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A Functional Hybrid Memristor Crossbar-Array/CMOS Systems for Data Storage and Neuromorphic Applications," *Nano Letters*, Vol. 12, No. 1, pp. 389-395, December 2011.

[20] Y.V. Pershin and M. Di Ventra, "Practical Approach to Programmable Analog Circuits with Memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 8, pp.1857-1864, August 2010.

[21] K. Tsutsui, "Focus on Strengths and Weaknesses of ReRAM," *Proceedings of the Flash Memory Summit*, August 2013.

[22] Z. Guz, I. Keidar, A. Kolodny, and U. C. Weiser, "Utilizing Shared Data in Chip Multiprocessors with the Nahalal Architecture," *Proceedings of the Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 1-10, June 2008.

[23] E. Schnarr and J. R. Larus, "Fast Out-Of-Order Processor Simulation Using Memoization," *Proceedings of the International Conference on Architectural Support of Programming Languages and Operationg Systems*, pp. 283-294, December 1998.

[24] D. B. Strukov and K. K. Likharev, "CMOL FPGA: a Reconfigurable Architecture for Hybrid Digital Circuits with Two-Terminal Nanodevices," *Nanotechnology*, Vol. 16, No. 6, pp. 888-900, June 2005.

[25] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinalli, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Mederios-Riberio, and R. S. Williams, "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic", *Nano Letters*, Vol. 9, No. 10, pp. 3640-3645, September 2009.

[26] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic," *Proceedings of the International Cellular Nanoscale Networks and their Applications*, pp. 1-6, August 2012.

[27] G. Snider, "Computing with Hysteretic Resistor Crossbars," *Applied Physics A: Materials Science and Processing*, Vol. 80, No. 6, pp. 1165-1172, March 2005.

[28] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive Switches Enable 'Stateful' Logic Operations via Material Implication," *Nature*, Vol. 464, pp. 873-876, April 2010.

[29] S. Shin, K. Kim, and S.-M. Kang, "Reconfigurable Stateful NOR Gate for Large-Scale Logic-Array Integrations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 58, No. 7, pp. 442-446, July 2011.

[30] E. Lehtonen, J. H. Poikonen, and M. Laiho, "Memristive Statful Logic", *Memristor Networks*, A. Adamatzky and L. O. Chua (Ed.), Springer International Publishing, Chapter pp. 603-623, 2014.

[31] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The Desired Memristor for Circuit Designers," *IEEE Circuits and Systems Magazine*, Vol. 13, No. 2, pp. 17-22, second quarter 2013.

[32] P. Sheridan, K.-H. Kim, S. Gaba, T. Chang, L. Chen, and W. Lu, "Device and SPICE Modeling of RRAM Devices," *Nanoscale*, Vol. 3, pp. 3833-3840, August 2011.

[33] D. B. Strukov and R. S. Williams, "Exponential Ionic Drift: Fast Switching and Low Volatility of Thin-Film Memristors," *Applied Physics A: Materials Science and Processing*, Vol. 94, No. 3, 515-519, March 2009.

[34] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching Dynamics in Titanium Dioxide Memristive Devices," *Journal of Applied Physics*, Vol. 106, No. 7, pp. 1-6, October 2009.

[35] W. Zhao, E. Belhaire, Q. Mistral, C. Chappert, V. Javerliac, B. Dieny, and E. Nicolle, "Macro-Model of Spin-Transfer Torque Based Magnetic Tunnel Junction Device for Hybrid Magnetic-CMOS Design," *Proceedings of the IEEE International Behavioral Modeling and Simulation Workshop*, pp. 40-43, September 2006.

[36] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, Vol. 18, No. 2, Part 2, pp. 210-214, June 2009.

[37] F. Corinto, and A. Ascoli, "A Boundary Condition-Based Approach to the Modeling of Memristor Nanostructures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 59, No. 11, pp. 2713-2726, November 2012.

[38] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A Versatile Memristor Model with Non-linear Dopant Kinetics," *IEEE Transactions on Electron Devices*, Vol. 58, No. 9, pp. 3099-3105, September 2011.

[39] S. Shin, K. Kim, and S.-M. Kang, "Compact Models for Memristors Based on Charge-Flux Constitutive Relationships," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 29, No. 4, pp. 590-598, April 2010.

[40] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211-221, January 2013.

[41] S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Models of Memristors for SPICE Simulations," *Proceedings of the IEEE Convention of Electrical and Electronics Engineers in Israel*, pp. 1-5, November 2012.

[42] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY Logic Design Flow," *Proceedings of the IEEE International Conference on Computer Design*, pp.142-147, October 2011.

[43] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI)* (in press).

[44] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC – Memristor Aided LoGIC," *IEEE Transactions on Circuits and Systems II: Express Briefs* (in review).

[45] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic," *Proceedings of the International Cellular Nanoscale Networks and their Applications*, pp. 1-6, August 2012.

[46] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL – Memristor Ratioed Logic for Hybrid CMOS-Memristor Circuits," *IEEE Transactions on Nanotechnology* (in review).

[47] Y. Levy, J. Bruk, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaacobi, and S. Kvatinsky, "Logic Operation in Memory Using a Memristive Akers Array," *Microelectronics Journal* (in review).

[48] R. Patel, S. Kvatinsky, E. G. Friedman, and A. Kolodny, "Multistate Register Based on Resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI)*, (in review).

[49] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Multithreading," *IEEE Computer Architecture Letters*, 2013 (in press).

[50] S. Kvatinsky, Y. H. Nacson, R. Patel, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "On the In-Die 3D Integration of Memory in CMOS Metal Layers and Its Implications on Processor Microarchitecture," submitted to *the Annual IEEE/ACM International Symposium on Microarchitecture*.

[51] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-Path Constraints in Memristor Crossbar Arrays," *Proceedings of the IEEE International Symposium on Information Theory*, pp. 156-160, July 2013.

[52] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "On the Channel Induced by Sneak-Path Errors in Memristor Arrays," *Proceedings of the International Conference on Signal Processing and Communication*, July 2014 (in press).

[53] A. Ascoli, F. Corinto, V. Senger, and R. Tetzlaff, "Memristor Model Comparison," *IEEE Circuits and Systems Magazine*, Vol. 13, No. 2, pp. 89-105, second quarter 2013.

מעגלים וארכיטקטורות מבוססי ממריסטור

שחר קוטינסקי

מעגלים וארכיטקטורות מבוססי ממריסטור

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר דוקטור לפילוסופיה

שחר קוטינסקי

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

סיוון תשע"ד חיפה מאי 2014

המחקר נעשה בהנחיית פרופ' אבינעם קולודני, פרופ' איבי פרידמן ופרופ' אורי וייזר בפקולטה להנדסת-חשמל.

אני מודה ל-Hasso Plattner Research Institute, אירווין וג'ואן ג'ייקובס, ולאנדרו וארנה פינצ'י ויטרבי על התמיכה הכספית הנדיבה בהשתלמותי.

תקציר

בשנת 1945 הציע המתמטיקאי פרופ' ג'ון פון נוימן מודל למבנה המחשב בו יחידת החישוב והבקרה (דהיינו המעבד) ויחידת אחסון המידע (הזיכרון) נפרדות זו מזו והעברת המידע נעשית באמצעות תקשורת בין הרכיבים השונים. מאז ועד ימים אלו, מחשבים נבנים על פי מודל זה, המכונה מכונת פון נוימן. למרות שמבנהו הבסיסי של המחשב לא השתנה, המשיכו ביצועי המחשב להשתפר. הגורם המרכזי לשיפור יכולות המחשבים בחמישים השנים האחרונות קשור בצורה חזקה למזעור הטרנזיסטורים בטכנולוגיית CMOS, שמהם מורכבים רכיבי החישוב והאחסון. לפי חוק מור, שגובש בשנת 1965 על-ידי ד"ר גורדון מור, מייסדה של חברת אינטל, מזעור הטרנזיסטורים יכפיל מדי כשנתיים את מספרם על גבי שבב בודד. המספר הגדל של טרנזיסטורים מאפשר להאיץ את ביצועיו של המחשב ולהגדיל את נפח אחסון המידע בו.

בשנים האחרונות יכולת מזעור הטרנזיסטורים הואטה ואילו יכולת העברת המידע בין הרכיבים השונים הפכה לצוואר בקבוק הן מבחינת ביצועי המערכת והן מבחינת צריכת ההספק. בנוסף, היות וצריכת ההספק הפכה לתכונה החשובה ביותר ביכולות המחשב, לא ניתן להמשיך ולנצל את יכולותיהם של כלל הרכיבים במערכת בו זמנית. בשנים הקרובות לא ניתן יהיה להמשיך ולהגדיל את נפח האחסון של טכנולוגיות הזיכרון הקיימות (זיכרון הבזק, DRAM ו-SRAM) תוך שמירה על צריכת הספק ומחיר נמוכים. טכנולוגיית זיכרון הבזק כבר הגיעה למימדים פיזיים של 15 ננומטר, שהם על פי מרבית התחזיות קצה גבול יכולת המזעור לטכנולוגיה זו, בעיקר בשל מגבלה על כמות הכתיבות לתא זיכרון בודד (endurance), אשר מחריפה ככל שהרכיב קטן יותר.

כדי להתמודד עם בעיות המזעור, מפותחות לאחרונה טכנולוגיות מוליכים למחצה המיועדות להחליף את טכנולוגיות הזיכרון הקיימות. בין הטכנולוגיות הללו ניתן לציין את Resistive RAM, MRAM-STT ו-PCM. טכנולוגיות אלו יכולות להיקרא בשם הכולל ממריסטור, על שם רכיב תיאורטי שנחזה בשנת 1971 על-ידי פרופ' לאון צ'ואה. ממריסטורים הם רכיבים בעלי שני הדקים, הדומים בפעולתם לנגד. התנגדותם של הממריסטורים משתנה כתלות בזרם הזורם דרכם, אך כאשר לא זורם זרם ההתנגדות נשארת קבועה ולכן הם רכיבים לא נדיפים (דהיינו לא דורשים מתח כדי לשמור על מצבם), בדומה לזכרונות הבזק. ההתנגדות של הממריסטור מייצגת את המידע האגור ברכיב. ממריסטורים מיוצרים בצפיפות גבוהה במיוחד בין שכבות מתכת, הנמצאות בכל טכנולוגיית CMOS סטנדרטית ולכן הם בעלי פוטנציאל גבוה למסחור ולשילוב במחשבים מודרניים כטכנולוגיות זיכרון חלופיות. ממריסטורים יכולים, בנוסף לאחסון המידע, לשמש גם כמתג ולהשתלב במעגלי CMOS שונים, בשימושים שונים ובהם מעגלים דיגיטליים, מעגלים אנלוגיים ורשתות נוירונים.

תמהיל התכונות של הטכנולוגיות הממריסטיביות הוא ייחודי ושונה מרכיבים חשמליים אחרים. ממריסטורים הם רכיבים קטנים וצפופים, המיוצרים מעל לטרנזיסטורים. הם מהירים, לא נדיפים וצורכים

הספק נמוך. תמהיל ייחודי זה מאפשר לממריסטורים להוסיף תכונות חדשות למעגלים חשמליים ופותח פתח למעגלים חשמליים חדשים ולארכיטקטורות מחשבים מגוונות.

הדגש במחקר זה הוא על השילוב בין זיכרון ליחידות החישוב במחשבים, בהתבסס על ממריסטורים. השילוב בין טכנולוגיית CMOS לבין טכנולוגיות ממריסטורים משפר משמעותית את המהירות וצריכת ההספק של מחשבים בעידן שבו המזעור לפי חוק מור ייגמר. שילוב בין ממריסטורים לטכנולוגיית CMOS מאפשר בניית ארכיטקטורות מחשבים חדשות המבוססות על שילוב כמות גדולה של זיכרון בתוך יחידות החישוב. לארכיטקטורות אלו אנו קוראים Memory Intensive Architectures.

במחקר זה פותחו מימושים שונים המבוססים על ממריסטורים, הן ברמת המעגל החשמלי והן ברמת הארכיטקטורה. המחקר כולל מספר רבדים שונים – רמת הרכיב הבודד, רמת המעגל החשמלי ורמת ארכיטקטורת המחשבים. במסגרת המחקר, אופיינו תכונות הממריסטורים הרצויות מנקודת מבטו של מתכנן המעגלים החשמליים וארכיטקט המחשבים. הממריסטור הרצוי שונה בהתנהגותו ובתכונותיו לשימושים השונים ולכן קיים צורך לאפיין היטב את מגוון הטכנולוגיות הממריסטיביות הקיימות ולפתח מודלים אשר יאפשרו את הגמישות הרצוייה עבור הרכיב. במסגרת המחקר, פותח מודל מתמטי בשם TEAM המתאר את התנהגות הממריסטור למגוון רחב של טכנולוגיות. מודל זה פשוט יחסית (דהיינו נדרש כוח חישוב נמוך יחסית כדי למדל אותו בסימולציות מחשב) ומדוייק. המודל מאפשר שימוש בזרם סף, שעבור זרמים הקטנים ממנו ההתנגדות של הרכיב לא משתנה, ובנוסף ממדל התנהגויות לינאריות ולא לינאריות של הרכיב. מודל ה-TEAM מומש בשפת VerilogA המותאמת לשפות תכן מעגלים חשמליים כגון SPICE.

מגוון שערים לוגיים מבוססי ממריסטורים פותחו במסגרת המחקר, כולל מתודולוגיות לתכנון יעיל ומדוייק שלהם. מתודולוגיות אלו כוללת שיטות לניתוח פעולת המעגל הלוגי, כולל בחינת החלופות השונות ודרכים לבחירת הרכיבים במעגל והפרמטרים השונים שלהם. שלוש משפחות לוגיות (IMPLY, MAGIC, Akers) ניתנות למימוש כחלק מזיכרון הבנוי מממריסטורים ולכן מאפשרות ארכיטקטורות השונות ממכונת פון נוימן ובהן חלק מהחישוב נעשה בתוך הזיכרון ולא ביחידת חישוב נפרדת. IMPLY ו-MAGIC ניתנות למימוש בתוך רשת ממריסטורים סטנדרטית המשמשת לזיכרון (crossbar). הפונקציה הבסיסית לחישובים ב-IMPLY היא פונקציית material implication וכתיבת הערך הלוגי 0 (אפס) ואילו ב-MAGIC הפונקצייה הבסיסית לחישובים היא פונקציית NOR. Akers היא רשת שונה, המבוססת על תאוריית חישוב לוגית שהוצעה ע"י פרופ' שלדון אייקרס בשנת 1972. שיטת החישוב של אייקרס מאפשרת לחשב כל פונקצייה בוליאנית, כולל מיון סיביות. במחקר זה פותחה חומרה הממשת רשת Akers. רשת אייקרס מבוססת ממריסטורים יכולה בשילוב עם טרנזיסטורי CMOS לשמש גם כזיכרון.

משפחה לוגית נוספת שמוצעת במחקר זה, MRL, מאפשרת שילוב עם שערים לוגיים בטכנולוגיית CMOS. עבור שיטה זו הממריסטורים משמשים כמתגים המסייעים לחישוב בלבד, ללא יכולת אגירת

מידע. השערים הבסיסיים הממומשים ב-MRL הם שערי AND ו-OR ובתוספת עם מהפכים מבוססי CMOS ניתן לחשב באמצעותם כל פונקצייה בוליאנית. היות וממריסטורים מיוצרים בין שכבות המתכת שמעל לטרנזיסטורי ה-CMOS, ניתן להגדיל את הצפיפות הכללית של השערים הלוגיים ובכך להמשיך ולשפר את יכולות החישוב של המחשב, גם ללא מזעור הטרנזיסטורים.

מעגל נוסף שפותח במסגרת המחקר הינו מעגל זיכרון בשם Multistate register. מעגל זה לא נועד להחליף טכנולוגיית זיכרון קיימת בהיררכיית הזיכרון של המחשב, אלא לשמש לצרכים חדשים ולשמירת מידע שבארכיטקטורות מחשבים קלאסיות לא נשמר. במעגל זה מאוחסנים מספר רב של מצבים, כאשר מצב אחד הינו פעיל ושאר המצבים נשמרים ברקע. עבור המצב הפעיל, משמש המעגל כרגיסטר רגיל בדומה לדלגלג CMOS. בעת הצורך, ניתן להחליף בין המצבים ולהפוך את אחד המצבים ששמורים ברקע למצב הפעיל. במחקר זה, מומש המעגל באמצעות ממריסטורים, כאשר המצבים השמורים ברקע נשמרים בתוך זיכרון מבוסס ממריסטורים והמצב הפעיל מאוחסן בדלגלג CMOS הממוקם מתחת לממריסטורים. תכנון זה מאפשר החלפה פשוטה ומהירה בין המצבים השונים, תוך אחסון מספר רב של מצבים שונים בשטח קטן. לדוגמה, עבור מעגל השומר בממריסטורים 64 מצבים שונים, כל מצב תופס שטח של כ-1.3% משטח מצב יחיד בדלגלג CMOS (צפיפות גבוהה פי 75 מדלגלג CMOS). הצפיפות היחסית של כל מצב משתפרת ככל ששומרים מספר רב יותר של מצבים.

השילוב של מעגל multistate register בתוך מעבדים יוצר אפשרויות לארכיטקטורות מעבדים חדשות כגון Continuous Flow Multithreading (CFMT). ב-CFMT רק תהליכון אחד פעיל בכל רגע נתון ושאר התהליכונים הנתמכים במכונה מאוחסנים בממריסטורים של מעגלי ה-multistate register. ההחלפה בין תהליכונים דורשת רק החלפת מצב פעיל במעגלי הזיכרון ולכן היא מהירה וחסכונית באנרגיה. מעבדי CFMT הם מעבדים בעלי ביצועים גבוהים וצריכה אנרגטית נמוכה. במסגרת הערכת הביצועים והאנרגיה של המעבד בסדרת בדיקות של SPEC CPU 2006 התקבל שיפור ביצועים ממוצע של 32%, תוך צריכה אנרגטית ממוצעת הנמוכה ב-8.5% יחסית למעבד דומה בו התהליכונים מוחלפים בשיטת Switch on Event Multithreading. במסגרת המחקר תוכננו מעבדי CFMT, כולל מימושם בחומרה אמיתית בטכנולוגיית CMOS על בסיס FPGA.

ארכיטקטורת CFMT הינה דוגמה ראשונה בלבד לארכיטקטורה המשלבת זיכרון מעל ללוגיקה (memory intensive architecture). בצורה דומה ניתן לתכנן ארכיטקטורות רבות נוספות. ארכיטקטורות אלו יכולות להתבסס על multistate registers, על מעגלי הלוגיקה שפותחו, או על מעגלים ממריסטיביים אחרים. אנו מאמינים שהמחקר המוצג הוא רק קצה הקרחון וכי בעתיד הקרוב ארכיטקטורות רבות נוספות ומימושים חדשים יפותחו, אשר ישנו את מבנה וארכיטקטורת המחשבים המודרניים.

לסיכום, התרומות המרכזיות במחקר זה הינן:

- ניתוח התנהגות הממריסטור והתכונות החדשות שהוא מוסיף למעגלים חשמליים ולמערכות מחשבים.
- פיתוח מודל ה-TEAM, המתאים לתוכנות תכן מעגלים חשמליים.
- פיתוח ומימוש מעגלי לוגיקה לחישוב בתוך הזיכרון ופיתוח מתודולוגיות תכנון עבורם (Akers ,MAGIC ,IMPLY).
- פיתוח ומימוש מעגלי לוגיקה המשולבים בטכנולוגית CMOS ופיתוח מתודולוגיות תכנון עבורם (MRL).
- פיתוח ומימוש מעגל זיכרון משולב ממריסטור ו-CMOS, המשמש לאחסון מספר רב של מצבים בשטח קטן (multistate register).
- פיתוח מיקרוארכיטקטורה למעבדים מרובי תהליכונים בעלת יחס גבוה של ביצועים לאנרגיה (CFMT).