The Dissertation Committee for Nagaraja Revanna Certifies that this is the approved version of the following dissertation:

Memristor Based Arithmetic Circuit Design

Committee:

Earl E Swartzlander, Jr., Supervisor

Jonathan Valvano

Andreas Gerstlauer

Deji Akinwande

Michael Schulte

Memristor Based Arithmetic Circuit Design

by

Nagaraja Revanna, B.E.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2016

Dedication

This work is dedicated to my parents, whose constant motivation and support through the years has been indispensable.

Acknowledgements

I would like to express gratitude to my advisor, Dr. Earl Swartzlander for his immense support and guidance throughout my Ph.D. His vast experience and insight has proven to be invaluable. His patience and encouragement really has no match. He has helped me evolve my thinking and instilled a sense of confidence to tackle new problems. I thank the dissertation committee members for serving on the committee and providing interesting suggestions. Their advice has helped improve the quality of my work significantly.

I am indebted to my parents, Leelavathi and Revanna Siddappa. I thank them for being my role models. Their encouragement and co-operation has been the impetus behind all my accomplishments. I thank all my teachers who have been a part of my education for the past 23 years. Their contributions at various stages of my life has helped in defining me. I thank my friends, family and colleagues for their encouragement.

A special thanks for Dr. Jonathan Valvano and Dr. Andreas Gerstlauer. I have learned a lot by working with them as a teaching assistant for over 2 years. I thank the many unnamed reviewers of different journals/conferences for their feedback on my work, which has helped address a lot of opens in the research. I thank Dr. Jack Lee, Dr. Mohamed Gouda, Dr. TR Viswanathan, Dr. T Lakshmi Viswanathan, Dr. Deji Akinwande, Dr. Michael Schulte, Prof. Eric Swanson, Jhonny Wong, Yannick Leclercq, Binta Patel and Susumu Hara for their support and guidance throughout my graduate studies.

Memristor Based Arithmetic Circuit Design

Nagaraja Revanna, Ph.D.

The University of Texas at Austin, 2016

Supervisor: Earl E. Swartzlander, Jr.

The revolution in electronics enabled by Moore's Law has been driven historically by the ability to fabricate ever smaller features lithographically on planar semiconductor platforms. In recent years, this has been slowing down due to the myriad of problems in short channel CMOS technologies. Research is now focusing on realizing Moore's law by architectural innovation, involving novel circuits and computation paradigms. There has been intense interest and activity directed towards designing logic circuits with memory elements. This is mainly driven by ideas like in-memory compute where logic operations are performed at the memory location in order to overcome the memory-wall bottleneck. Resistive-switching random-access memory (RRAM)/ memristors has a great potential to be the future of non-volatile memory owing to its CMOS compatibility, read-write endurance, power and speed. We describe novel high speed logic circuits for adders and multipliers built with RRAM to support the concept of logic-in-memory. These circuits have significant speed/area/power improvements over the existing designs. The complexity involved in computation in terms of controlling the basic gates, sequence of operations etc. has been significantly reduced. RRAM properties are exploited with the help of a well-known analog element called current mirror. Previously known logic-implication technique to realize digital gates comes with a serious limitation of limited fan-out. By using current mirrors, this limitation can be overcome, enabling more logic operations to

run in parallel. Results show that the delay for even an XOR operation can be reduced to 1 cycle, compared to the 5 cycles taken by logic implication. Spice simulations are done with known RRAM models. Simulation results show significant improvement in power consumed over the existing designs. The design of different adders and multipliers are also described. Metrics like area, power and latency are compared, and it shows significant improvement over the state-of-the-art.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 MOTIVATION

The concept of memristors was first hypothesized by Leon Chua [1] in 1971 as a two-terminal device, which establishes a fundamental relation between magnetic flux ϕ and charge q as shown in Figure 1. The relation for a memristor with memristance M is given by $d\phi = M(q)\, dq$. Memristor is a contraction of "memory resistor," because it behaves much like a resistor at a given time, with the resistance depending on the past history of the current passing through it. Memristors were realized by HP in 2008 in nanoscale titanium-di-oxide cross-point switches [2]. Since then, much research has been done in exploring the use of memristors as non-volatile memory storage elements [3], called Resistive RAM (ReRAM). There has been some work in using memristors to build logic circuits [4, 5].

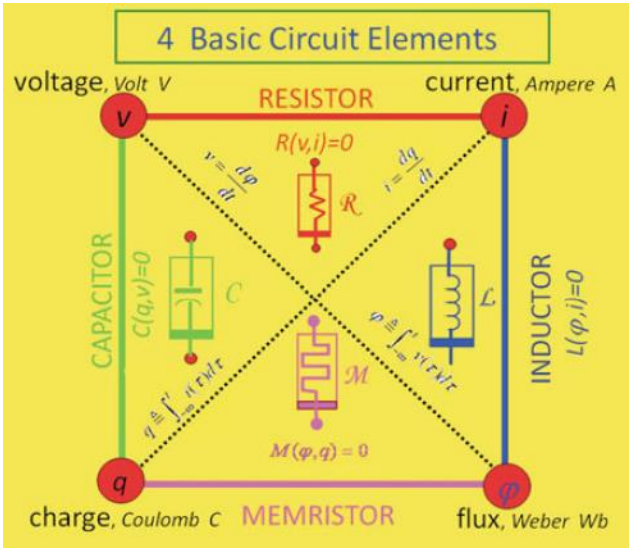


Figure 1: Memristor hypothesized as the 4th fundamental element [6]

Memristors are very reliable memory elements [7,8]. The have better endurance than conventional flash or FeRAM. The write/erase time is in the order of 0.1ns, comparable to conventional DRAM. Their ability to retain memory even when powered down has made memristors a very attractive choice and has received significant attention from the research community to develop and improve the process of fabricating memristors. Many applications like an SOC of mobile systems, sensor nodes in IoT etc. implement a "Race to Sleep" type architecture where the systems are put to sleep when not in use to conserve battery power. The DRAM in these systems needs constant refreshing to retain data, which hits hard on the power budget. Memristors provide a promising solution to this problem.

The revolution in electronics enabled by Moore's Law has been driven historically by the ability to fabricate ever smaller features lithographically on planar semiconductor platforms. In recent years, this has been slowing down due to the myriad of problems in short channel CMOS technologies. Research is now focusing on realizing Moore's law by architectural innovation, involving novel circuits and computation paradigms. There has been intense interest and activity directed towards improving computation by parallelism. The motivation for designing logic circuits with memory elements is mainly driven by applications like in-memory compute.

In the conventional Von Neumann model, computing systems are physically and logically split between memory and CPUs. This implies that data must be moved between storage and processor. The lower speed of the memory relative to the processor results in the memory wall, which has severely limits the efficiency of many applications. When running today's memory intensive applications, modern computers spend most of their time and energy moving data rather than on computation, wasting energy. The idea of

fetching data from the memory and processing is in the CPU is what is limiting the performance of the computer.

Logic-in-memory designs tightly integrates specialized computation logic with embedded memory, enabling more localized computation, thus save energy consumption. The realization of logic operations within passive crossbar memory arrays is a promising approach to expand the fields of application of such architectures. There is a push to have a three-dimensional integration of active semiconductor devices and memory. This trend has been particularly prominent in development of nonvolatile memory technology. Nonvolatile electronic (flash) memories are based primarily on floating-gate structures, invented in 1967 by D. Kahng and S. M. Sze, and are ubiquitous in portable electronic products such as mobile phones, digital cameras, notebook computers, mp3 players and USB flash drives.

Memristors help alleviate this scenario by allowing the use of a memristor as both computation and memory element. Memristor-based implication 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 massively parallel applications, where vast amount of data need to be processed. 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 [9].

Memory	Flash		FeRAM	MRAM	PCRAM	RRAM
Attributes	NOR	NAND	Prototypical products			
Knowledge level	Mature					
Cell elements	1T		1T1C	1T1R	1T1R	1T1R/1D1R
Cell area (F^2)	10	5	22	20	4.8	4[a]
Write/Erase time	1us/10ms	1ms/100us	10ns/10ns	20ns/20ns	50ns/120ns	0.3ns[b]
Write voltage (V)	12	15	0.9-3.3	1.5	3	<3
Endurance	10^6		10^{12}	$>10^{14}$	10^{12}	10^{10}
Scalability	Major technological barriers		poor	poor	Moderate	Excellent

Figure 2: RRAM/Memristor against other non-volatile memory technologies.

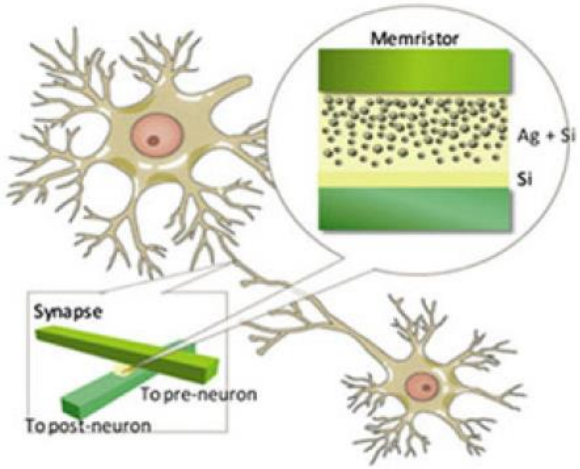


Figure 3: Memristor in Neuromorphic computing [10]

Figure 2 shows a comparison between different non-volatile memories. It shows that RRAM has very low write voltages with relatively good endurance and excellent scalability and compatibility with CMOS. This is a good proof that RRAM can be the potential candidate for the future of non-volatile RAM. Researchers are trying to mimic the signal processing in the brains of mammals. In such computations, the degree of

complexity increases exponentially with traditional computing architectures. This necessitates the need for artificial neural networks and neuromorphic computing [10]. EPFL's Blue Brain project, IBM/DARPA's SyNAPSE, etc. are significant projects exploring neuromorphic computing. There is a huge scope for research in using memristors for logic applications. Figure 3 shows memristors being used as synapses if neurons in neuromorphic computing mainly because of the hysteresis properties of memristors.

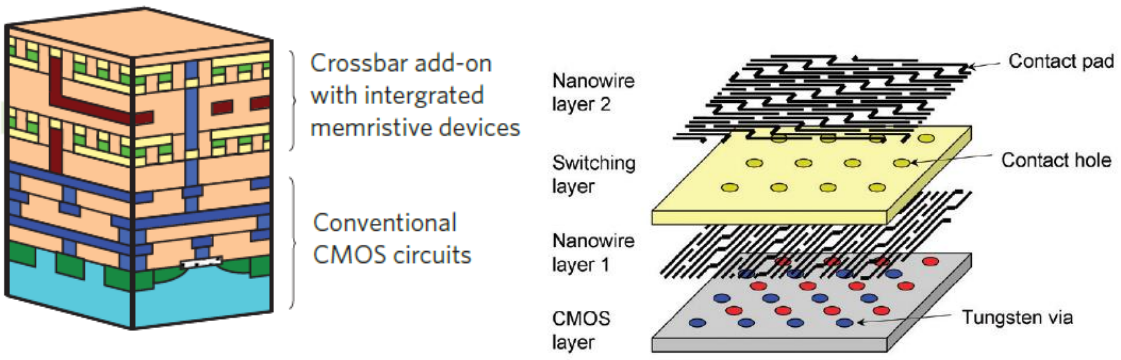


Figure 4: CMOS-Memristor integration [11].

Figure 4 shows an application where CMOS and RRAM cross bar matrix work hand-in-hand. In this application, the memristor cross points were used to make/break connections between different CMOS gates. Thereby, the gates can be programmed to have different connections dynamically. Memristor-based implication 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. Recently, there has been multiple papers exploring the benefits of logic-in-memory applications [12,13] as shown in Figure 5. Results shows that there is significant improvement in energy efficiency for accelerating memory-bound problems. The implementations though are very application specific. This also requires a change in the

applications algorithms. There has been other work exploring logic operations with nano magnets, focusing on enhancing logic-in-memory applications.

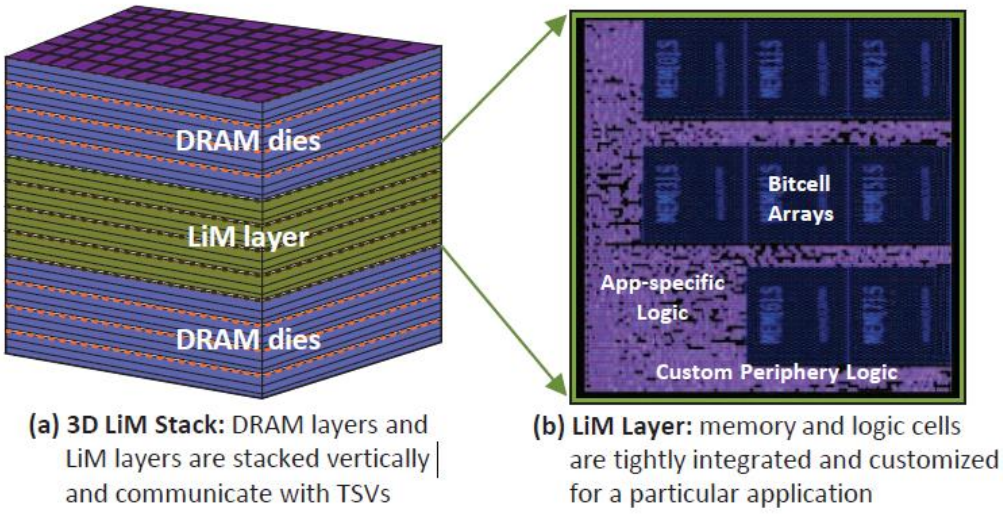


Figure 5: A 3D stacked Memory-logic-memory sandwich [12].

1.2 Approach and Methodology

This research work focuses on the design of memristor based arithmetic circuits like adders and multipliers. By using the implication operation described in section 2.3, memristor based logic gates can be designed. Based on this, multiple fixed point adder architectures like ripple carry adder, carry look-ahead adder, conditional sum adder, carry select adder, parallel prefix adders etc. are implemented. It is shown that the methodology adopted to design fixed point adders has resulted in better metrics than the state of the art designs employing implication logic [4,5,14-18]. With this, some of the shortcomings of implication logic is addressed in Chapter 4, where new circuits capable of having high fan-

out is explained. Simulations are done to prove the working of the concept with Verilog-A models of well-known memristors and the results are compared with published literature.

1.3 Structure of the Dissertation

The dissertation is divided as follows. The second chapter explains the basics of operation of a memristor and some of the initial circuit/logic design work done using memristors. The third chapter explains the design of logic gates with the concept of logic implication. Chapter 4 describes an entirely new concept of building high fan-out logic gates using memristors and current mirrors. The last chapter summarized the contributions and provides a direction for future work. An appendix with the Verilog-A model of the memristor used for simulations and a list of references is provided at the end of the dissertation.

1.4 Research Statement

RRAM/memristor based logic design is in its nascent stage. There is a huge potential for RRAM to be the future of non-volatile memory. The need to overcome the limitations in Von-Neumann computation methodology, logic-in-memory is a potential solution. This research explores and addresses some of the key issues involved in RRAM based logic circuit design. This research provides new methodologies for building adders and multipliers with the well-known imply-based operations. It makes used of parallelism, resource reuse and removes redundant terms to make the operation faster, low power and smaller. Furthermore, to overcome the inherent fan-out limitations in imply-based logic, this research presents a new approach which makes use of the fundamental principles of RRAM operation and used current mirrors to build logic gates. It has been shown that the

current mirror based gates is very CMOS friendly. It has better power, delay and area metrics among the state-of-the-art. The research identifies some of the key issues in current mirror based logic and provides opportunities for future exploration and development.

Chapter 2: Previous Work

The search for the ideal non-volatile memory has pushed the scientific community to design novel memory structures. Resistive RAM is one such area which has seen significant progress since the 2000s. It has great potential to be the future of non-volatile memory. This chapter explains some of the earlier works done in the area of memristor design and fabrication. It also explains some of the circuit design works using memristors in the recent past. It is interesting to note that memristors are being explored in a variety of applications, not just as memory. Applications like the well-known differential amplifier, digital logic gates, neuromorphic computing to mimic the action of neurons etc.

2.1 The First Memristor by HP

Memristive behavior was first observed by HP in 2008 in a Titanium – Titanium dioxide (TiO_2) cross point switch as shown in Figure 6. It is a sandwich of TiO_2 between 2 titanium electrodes/plates. The conductivity of the structure is explained through ion transport. The key to the switching was the formation of a bilayer of the two different titanium dioxide species. The TiO_2 is electrically insulating (actually a semiconductor), but the TiO_{2-x} is conductive, because its oxygen vacancies are donors of electrons, which makes the vacancies themselves positively charged. The operation is very simple. If a positive voltage is applied to the top electrode of the device, it will repel the (also positive) oxygen vacancies in the TiO_{2-x} layer down into the pure TiO_2 layer. That turns the TiO_2 layer into TiO_{2-x} and makes it conductive, thus turning the device on. A negative voltage has the opposite effect: the vacancies are attracted upward and back out of the TiO_2, and thus the thickness of the TiO_2 layer increases and the device turns off. [19]

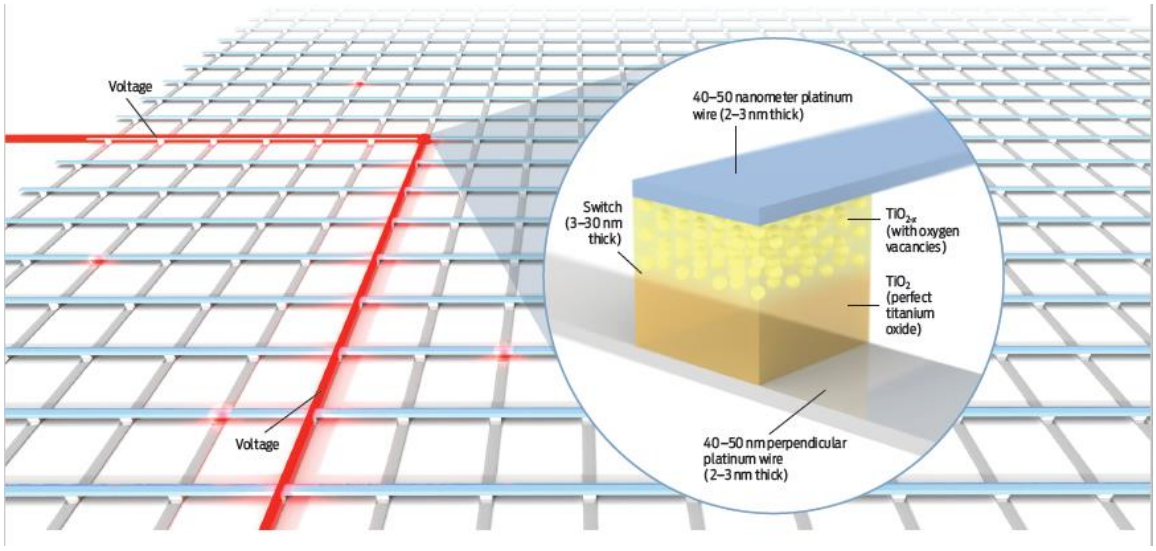


Figure 6: Structure of a memristor [19].

This effect can be clearly observed in the I-V characteristic plots of Figure 8. The following should be noted when working with memristors. The change of the memristive state depends on the voltage applied, the current limitation (also called compliance current) and the duration for which the voltage is applied. In Figure 7, it can be noted that when a sine wave is provided as an input, the current is not a pure sine wave. The resistance keeps changing as the voltage changes, causing a non-linear response in current.

The hysteresis curve clearly depicts the non-linear switching behavior where the memristor changes states from high resistance to low resistance. The window of hysteresis becomes narrower as the frequency of the applied switching voltage increases. The HP memristors were laid out in a crossbar architecture. The area of each junction/memristor was 100nm x 100nm. The integration with CMOS process was less complicated (not easy though) because the memristive operation was only in the metal layers. A hybrid FPGA was also fabricated by the HP team [11] to demonstrate the possibility of integration of standard CMOS and memristor materials. Hybrid reconfigurable logic circuits were

fabricated by integrating memristor-based crossbars onto a foundry-built CMOS platform using nanoimprint lithography, as well as materials and processes that were compatible with the CMOS. Titanium dioxide thin-film memristors served as the configuration bits and switches in a data routing network and were connected to gate-level CMOS components that acted as logic elements, in a manner similar to a field programmable gate array [11].

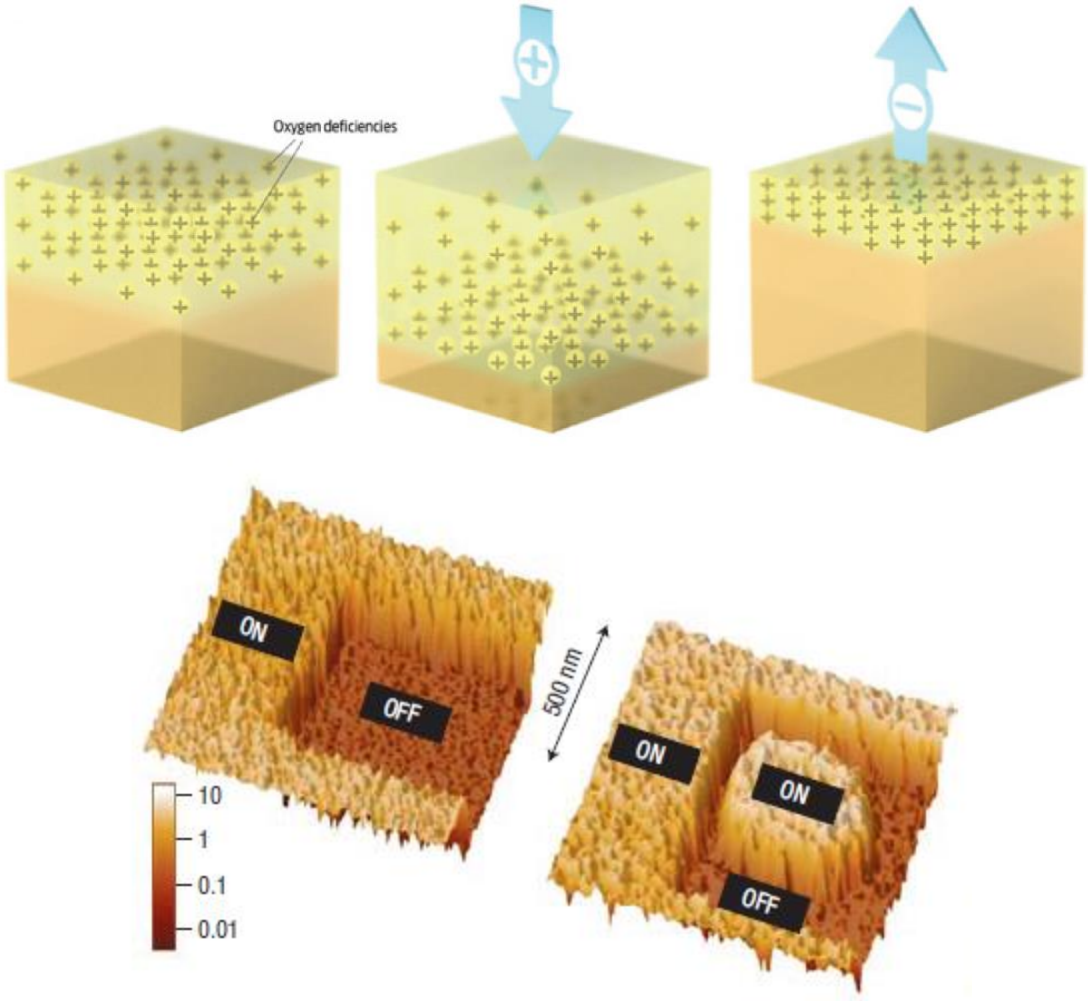


Figure 7: Ion motion in Ti – TiO_2 junctions [19] and in $SrTiO_3$ thin film materials [20].

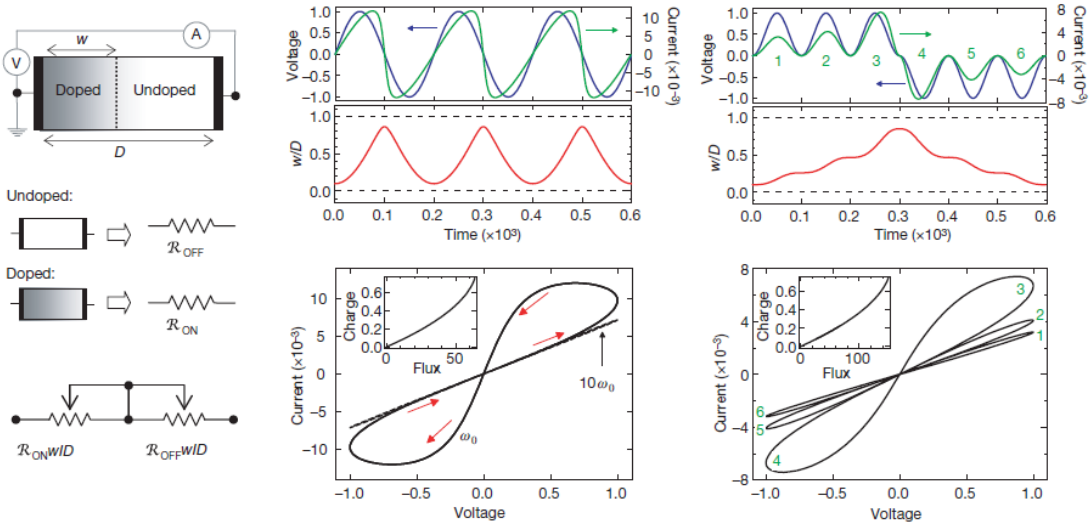
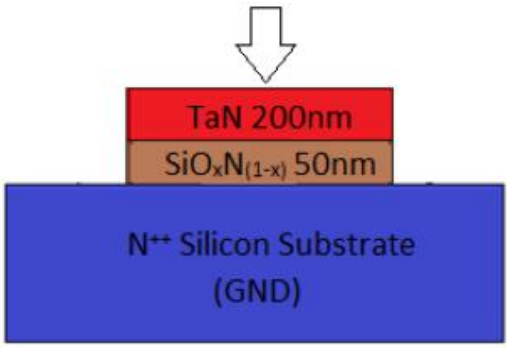


Figure 8: Memristor operation, I-V characteristics [2].

Recently, the property of memristors has been identified with standard CMOS with SiO_x as the gate dielectric and TaN based gate materials [7,8,21]. Figure 9 shows the typical structure of a CMOS based memristor and the I-V characteristics of the device. This example has a multilevel property, where the resistance depends not only on the applied voltage, but also on the compliance current. This work is worth mentioning because of the fact that there is no need for additional nanoimprint lithography like in [11] to integrate memristors in a standard CMOS process.



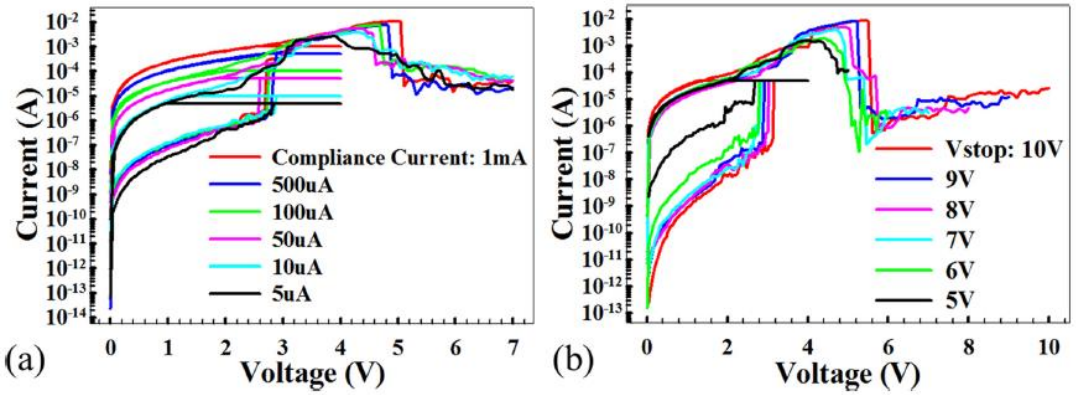


Figure 9: CMOS based memristor implementation and the I-V plots [7].

The switching is dominated by the conduction channels in the switching matrix materials. In Figure 10, a-d represents transport mechanisms of ions and electrons for the switching on anion-based devices. It depicts a simplified schematic of conduction channels (red) in switching matrix materials (blue) in four typical switching devices, where both electric field and Joule heating drive the switching. Different from **c**, the channel in **d** usually completely disappears in the high-resistance state and may or may not (for example, Mott transition) involve ionic motion. The possible electron transport paths in the devices are shown in Figure 10. Some of the mechanisms are as follows. [22]

- Schottky emission: thermally activated electrons injected over the barrier into the conduction band.
- Fowler–Nordheim tunnelling: electrons tunnel from the cathode into the conduction band; usually occurs at high electric field.
- Direct tunnelling: electrons tunnel from cathode to anode directly; only when the oxide is thin enough. When the insulator has localized states (traps) caused by disorder, off-stoichiometry or impurities, trap-assisted transport contributes to additional conduction, including the following steps:

- Tunnelling from cathode to traps.
- Emission from traps to the conduction band (Poole–Frenkel emission).
- Tunnelling from trap to conduction band
- Trap-to-trap hopping or tunnelling, ranging from Mott hopping between localized states to metallic conduction through extended states
- Tunnelling from traps to anode.

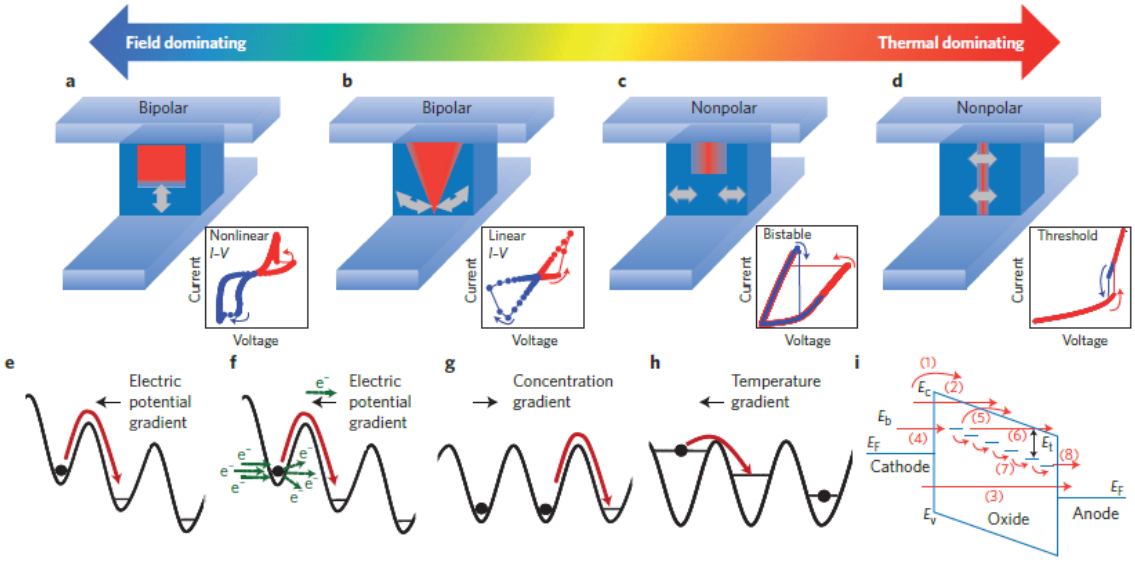


Figure 10: Switching in anion-based memristive devices [22].

2.2 MEMRISTOR CLASSIFICATION

Resistive random-access memory (RRAM) devices generally are simple in structure (typically two-terminal) and nanoscale in dimensions (scaling <10 nm has been demonstrated), while at the same time offering excellent performance in terms of switching speed and write/erase cycling. A large amount of work has been performed to understand the various types of switching mechanisms that are responsible for RS phenomena in

different material systems. They have been broadly categorized into valence change, electrochemical metallization, phase change, thermo-chemical, ferroelectric, and nanomechanical effects, as shown in Figure 11. Experimentally, devices that fall in the first three categories (namely, valence change, electrochemical metallization, and phase change effects) have been extensively studied for the purpose of neuromorphic computing.

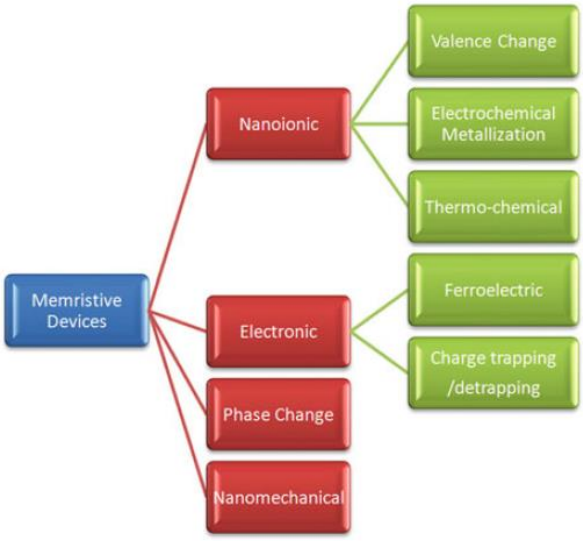


Figure 11: Classification of memristors based on switching mechanism.

2.3 LOGIC IMPLICATION

Memristors are typically used for memory applications. The use of them in logic gates was first demonstrated in [3] by an operation called implication. The I-V characteristics of the memristor in this application is shown in Figure 12. V_{CLEAR} is the voltage, which switches the memristor to a high resistance state (open) and V_{SET} is the voltage, which switches it to a low resistance state (close). V_{COND} is a voltage very close

to V_{SET}, but not enough to change the state of the memristor. The memristors P and Q are driven by buffers, which provides the appropriate voltages for the implication operation.

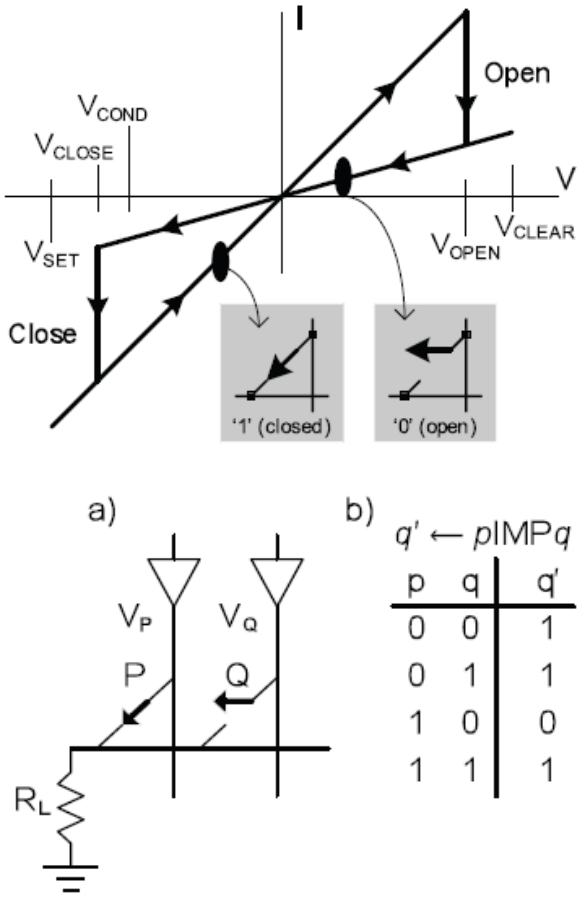


Figure 12: Implication operation using memristors [4].

An implication operation is denoted by $q^+ \leftarrow$ p imp q. The equivalent logic operation turns out to be $q^+ = p' + q$. For this operation, V_P is set at V_{COND} and V_Q at V_{SET}. If the memristor P is closed (P = 1), then V_P appears across R_L. The voltage across the memristor Q (which is now $V_{SET} - V_{COND}$) is not enough to change the state. Hence, Q retains its original state. If P is open (P = 0), then V_P is isolated from the resistor. The resistor is at a value, which limits the compliance current. Since V_Q is V_{SET}, it changes the

state of Q to "closed" (Q = 1). Thus, when P = 1, Q remains unchanged. When P = 0, Q is set to 1.

2.4 Previous Works with Memristors

Much work has been done using memristors for various applications including analog, memory, logic gates, hybrid FPGAs, neuromorphic computing etc. This section lists some of the more significant works.

2.4.1 Memory

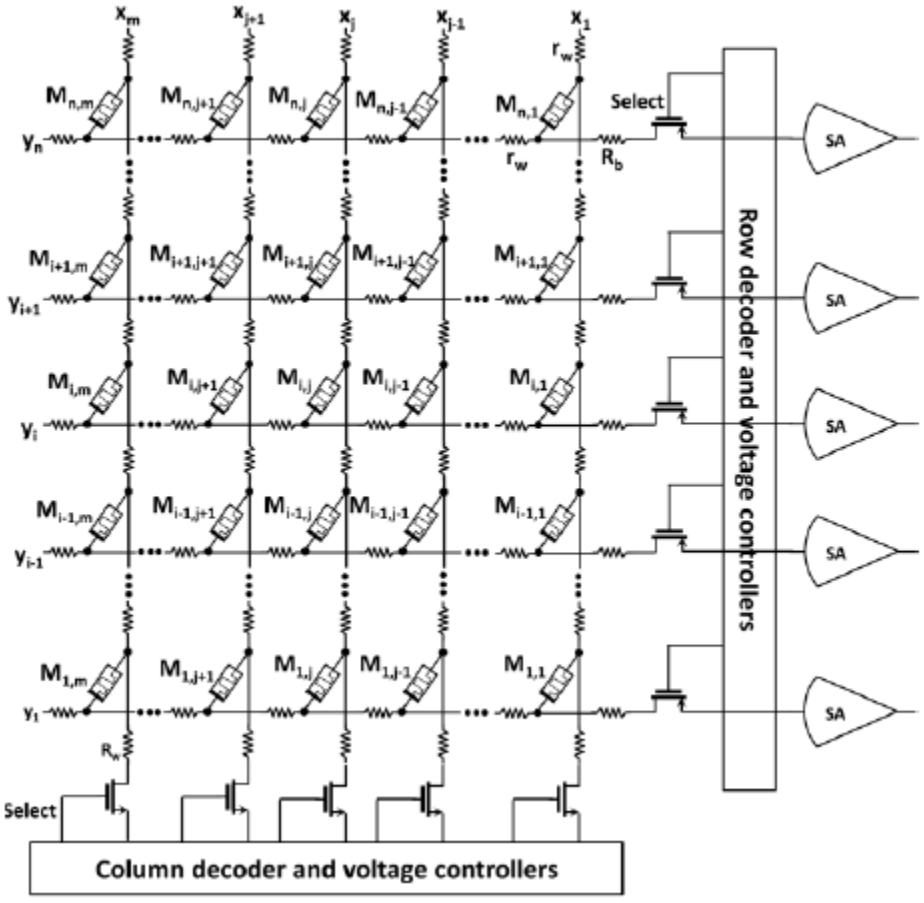


Figure 13: Analog arithmetic operations using memristors.

The fundamental application of memristors is memory. There has been significant amount of work to demonstrate the cross-bar control network, sense amplifier design etc. for a memristor crossbar. The structure is very similar to that of a DRAM crossbar matrix.

2.4.2 Linear Differential Amplifiers

Reference [23] is a work on using memristors to obtain highly linear differential amplifiers. The paper approximates the current through a memristor as a hyperbolic sine function. Since the differential amplifier has a hyperbolic tan – type non linearity, the sinh function extends the linear range of the amplifier. A main problem in this approach is that the hyperbolic sine function is a strong function of the process. Any process changes will result in bad harmonic content at the output of the differential amplifier.

2.4.3 Analog Memory

References [4,16,24] explain the use of memristors as analog memory. An "analog memory" is defined as a connected pair of memristors. This memristor pairing allows for the representation of both positive and negative values. The conductance of the top memristor, g_{mt}, of the pair is always programmed to half the conductance range. The conductance of the bottom memristor, g_{mb}, may be programmed within the full conductance range. If $g_{mb} < g_{mt}$ then the value of the analog memory is negative. If $g_{mb} > g_{mt}$ then the value of the analog memory is positive. Using analog memories, addition as well as multiplication by -1 can be performed.

2.4.4 Basic Analog Operations

There has been some work describing the use of memristors for addition/subtraction/multiplication and division operations. These are in the analog domain. Memristors are being treated as normal resistors. Although these ideas are feasible, it is very hard to control the resistance of a memristor. The variation of resistance is a

function of process, bias current, applied voltage and duration of the applied voltage. With all these variables, the error rate is prone to be very high in using memristors for precise analog applications. Research needs to be done on improving the quality of memristors, and to have a better and well defined control in setting the resistance of a memristor.

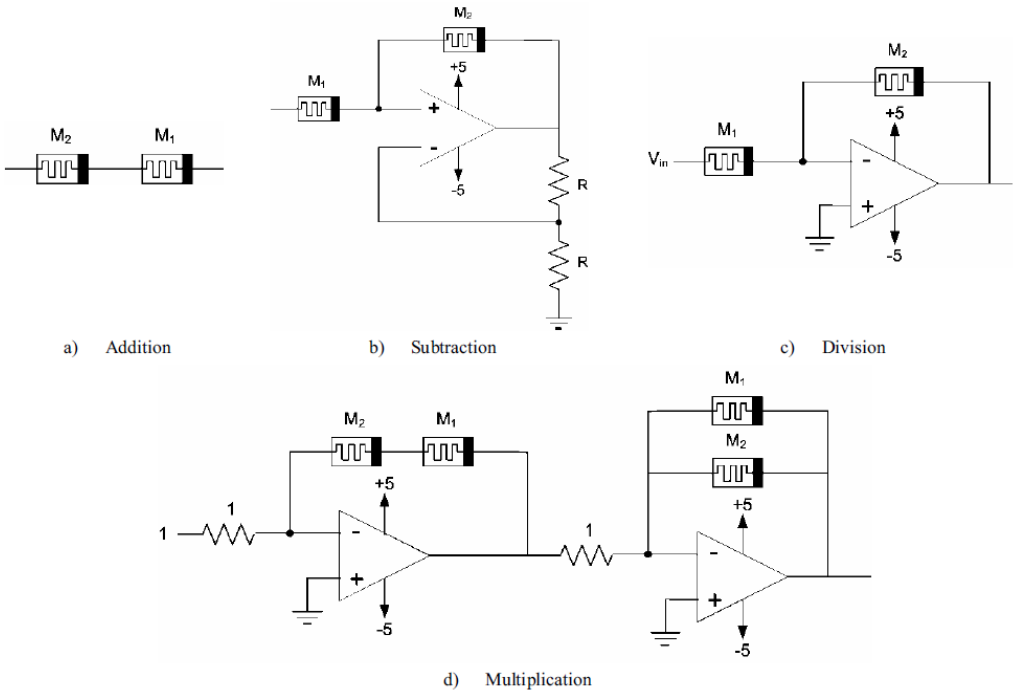


Figure 14: Analog arithmetic operations using memristors.

2.4.5 Security Applications

References [20,25] explain the use of memristors in approximate computing and security applications. Silicon PUFs (Physical Unclonable Functions) are on-chip circuitry that can extract fabrication variations, such as the discrepancy from the expected delay or the instable power-on states of the memory cells, to generate chip-dependent PUF data that can be used as secret keys or as seeds of random number generators, and/or to create challenge response pairs (CRPs) for authentication and attestation. These capabilities have

opened up new avenues for implementing hardware intrinsic security and trust. The memristor system state variable w(t) or the corresponding effective resistance M(w,i), provides an ideal situation to build memristor PUFs [20].

2.4.6 Digital Logic Applications

There has been some early work in [4,5] in realizing memristor based logic gates. These have very large delay numbers and hence the power consumed is significantly higher. [26] uses an improved technique, where memristors are placed in series or parallel to determine the outcome. A series connection results in a AND operation and a parallel connection results in an OR operation. But, this heavily relies on the crossbar matrix. It significantly slows down the amount of parallel operations that can run on the matrix at a given time.

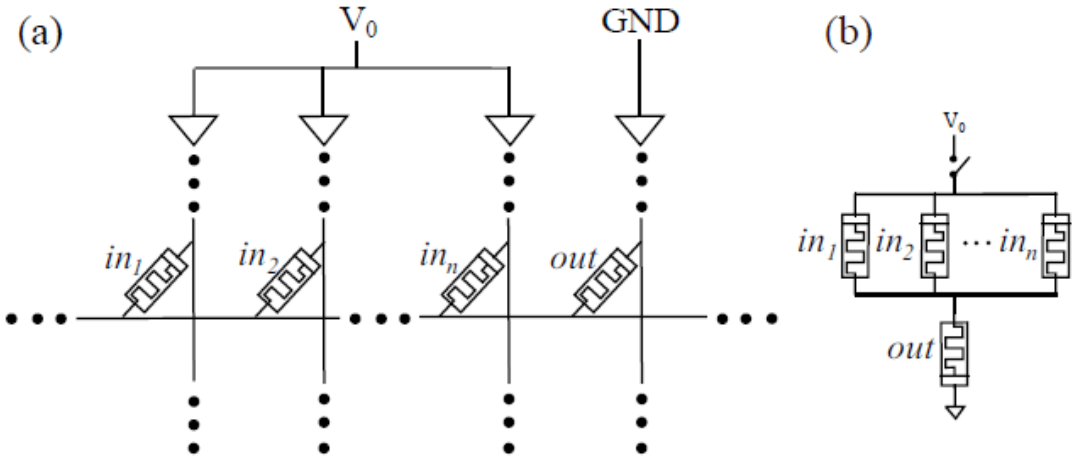


Figure 15: a) Part of the crossbar Matrix b) MAGIC based NOR gate.

Memristor as Driver (MAD) gates proposed in [27] relies on controlling switches depending on the voltage generated on the memristors. Figure 16 shows a typical OR and XOR gate realized as MAD gates. This technique addresses the issue of parallelism but has serious limitations when it comes to the design of the threshold switches.

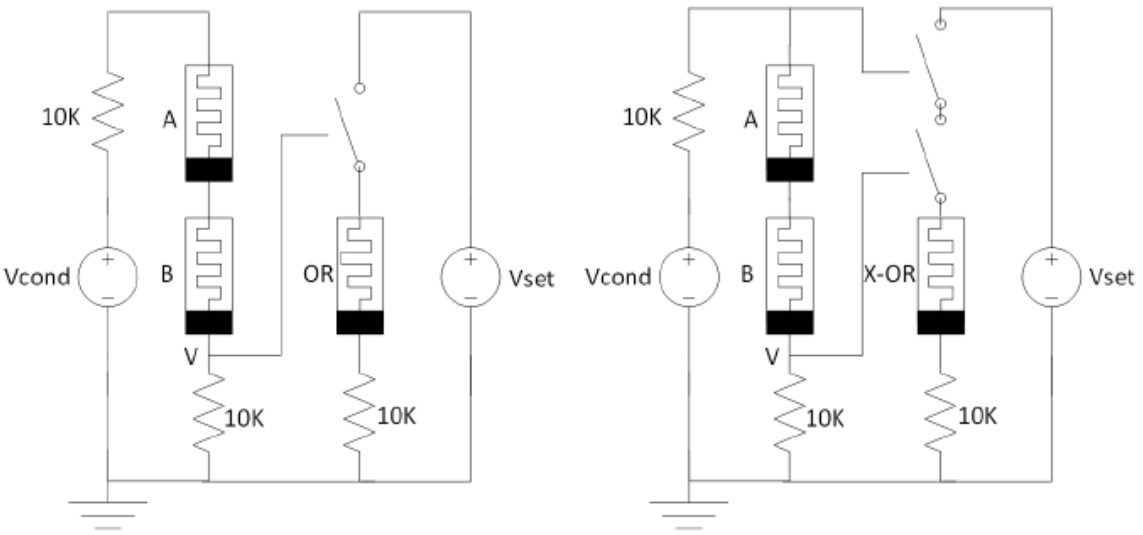


Figure 16: MAD based OR and XOR gates.

More work related to memristor based circuits is listed in the references [41 – 50].

2.5 MEMRISTOR DEVICE MODELS

A nonlinear device model consists of a set of ideal circuit elements appropriately connected together that can replicate the experimentally measured electrical properties of the device to a reasonable accuracy. Leon Chua constructed a circuit-element-array [28], also called a doubly periodic table of ideal circuit elements that covers the mathematically possible relationships between current and voltage in a two terminal device. It has also been proven that these 4 elements are independent of each other, in other words, fundamental. Chua has described a set of five qualities that a realistic device model should possess [28].

- It must be well-posed, or in other words it should not yield any nonphysical solutions, such as a voltage or current that tends to infinity in a finite time.

- It must have simulation capability, i.e., a computational solution of the model should yield results that are judged to be close enough to experimentally measured data for the physical device.
- The model should have qualitative similarity to the device, especially with respect to any limiting or asymptotic behaviors that are observed.
- A truly useful model must have predictive ability, in that simulations of operating environments for which no previous experimental data have been collected prove to be accurate when corresponding measurements are performed. The structural stability of the model means that its properties do not change qualitatively when small changes are made to the parameters of the model.
- An additional requirement is that any parameters in the model determined by fitting to experimental data should be intrinsic to the device and not depend on the external measurement circuitry.

Finally, there is the issue of judgment—if a model appears to violate a law of physics or involves parameters that diverge or oscillate wildly while the voltage and current of the physical device are well behaved, the model is not useful and a different model needs to be formulated, likely using a different set of ideal circuit elements.

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 must 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. [29]

2.5.1 Linear ION Drift Model

The linear ION drift model is described in [2]. It was described mainly to model the memristor formed by the Titanium – TiO_2 junction. In this model, the assumption is that a device of physical width contains two regions, as shown in Figure 8. One region, which acts as the state variable of the system has a high concentration of oxygen deficient spots or dopants (oxygen vacancies of TiO_2, namely TiO_{2-x}). The second region of width is an oxide region (TiO_2). The region with the dopants/oxygen deficiency has a higher conductance than the oxide region, and the device is modeled as two resistors connected in series. Several assumptions are made in this model. It assumes that the conductance is ohmic. Ion drift is linear in a uniform field, and that the ions have equal average ion mobility. Here, w(t) is the state variable and is limited to the physical dimensions of the device.

$$\begin{aligned} \frac{dw}{dt} &= \mu_v \frac{R_{ON}}{D} i(t), \\ v(t) &= \left(R_{ON} \frac{w(t)}{D} + R_{OFF} \left(1 - \frac{w(t)}{D} \right) \right) \cdot i(t) \end{aligned} \tag{1}$$

Sometimes, a window function is used to limit w(t). There are several window functions like Biolek Window, Jogelkar Window, Prodromakis Window etc. The above equation is multiplied by a function that nullifies the derivative and forces the equation to be 0 when w is at bound. “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 reduces as shown in Figure 17. This is described in equation 2.

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

$$f(w) = j\left(1 - \left[\left(\frac{w}{D} - 0.5\right)^2 + 0.75\right]^p\right) \tag{3}$$

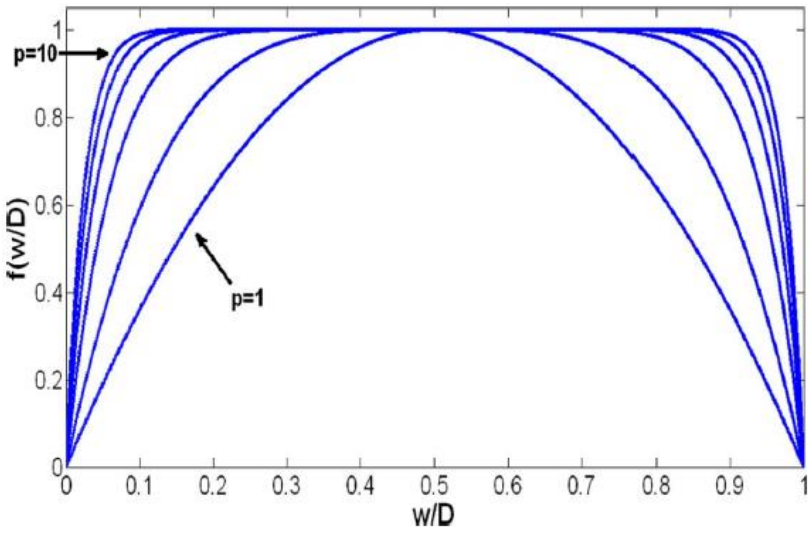


Figure 17: Variation of the window function f(w) as a function of *p*.

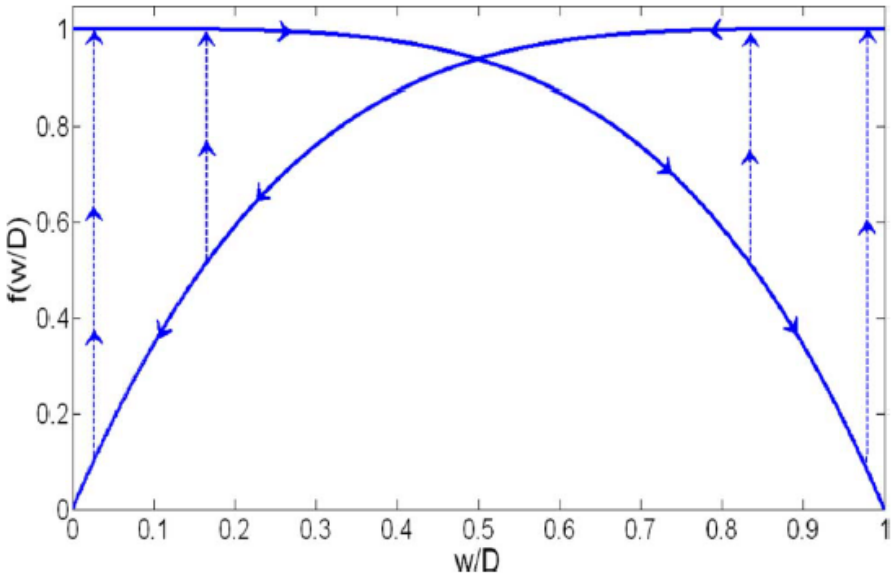


Figure 18: Biolek Window Response.

The Prodromakis window function is described by equation 3. Here, "j" is a control parameter which determines the maximum value of f(w). This response is shown in Figure 19.

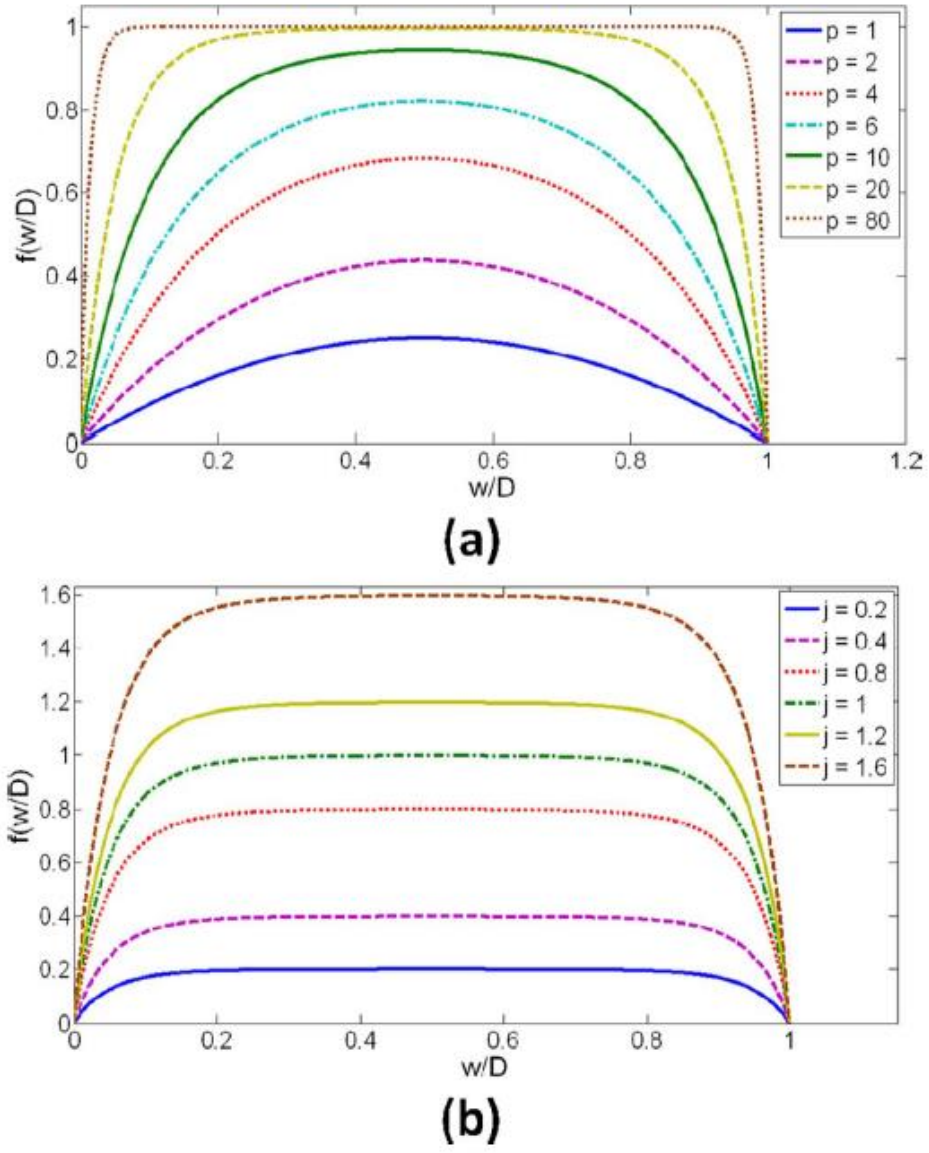


Figure 19: Prodromakis Window response a) Varying *p* b) Varying *j*.

2.5.2 Non-linear ION Drift Model

Linear ion drift model 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. The nonlinear I-V characteristic is desirable for logic circuits, and hence more appropriate memristive device models have been proposed. Figure 20 shows the response of non-linear ION drift modeling of a memristor.

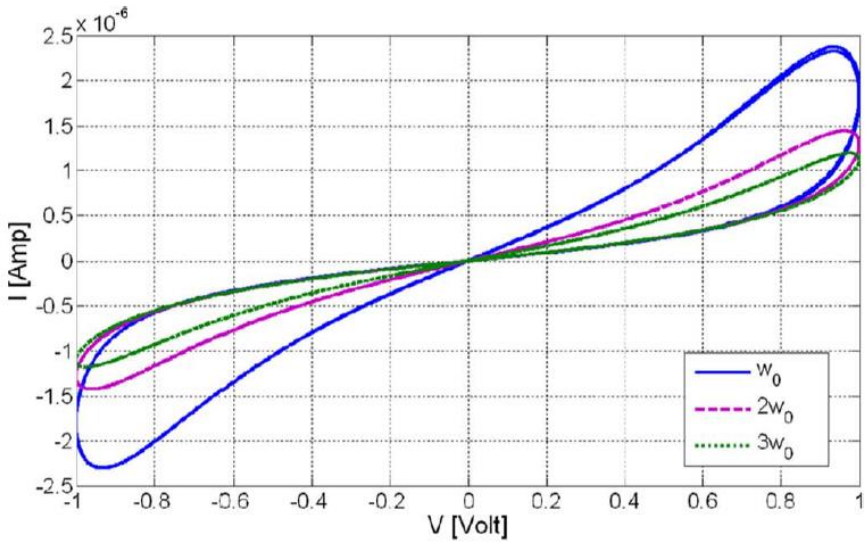


Figure 20: Non-linear ION drift modeling, I-V characteristics with frequency.

2.5.3 Simmons Tunnel Barrier Model

Linear and Non-linear ION models are based on modeling the 2 regions of oxide and doped oxide as 2 resistors in series. A more accurate model was proposed in [30] which assumes assymetric switching behavior due to the exponential dependence of the movement of the ION dopants. The physical model of Simmons tunnel barrier model is shown in Figure 21. This shows a physical resistor R_s in series with an electron tunnel

barrier. “x” is the state variable which is also the tunnel barrier width. Figure 22 captures the variation of the tunnel barrier in a Ti-TiO_2 junction. It shows the variation of width with time across different values of applied voltage. The second plot provides the variation of the derivative.

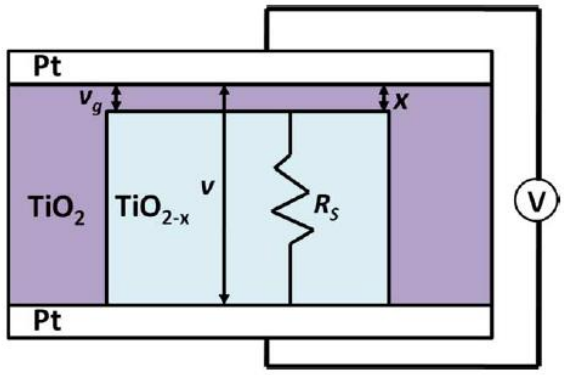


Figure 21: Physical model of Simmons Tunnel Barrier model.

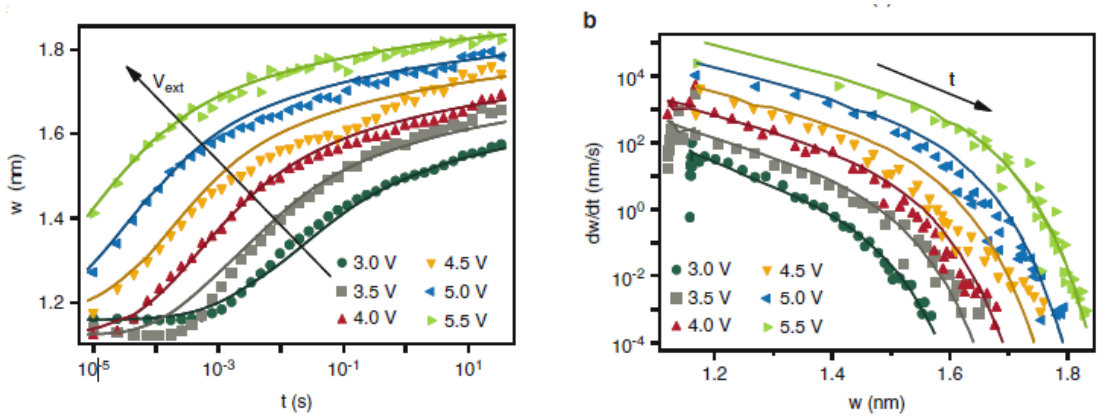


Figure 22: Variation of “w” and its derivative with time and applied voltage.

The equations which describe this model is given in equation (4).

$$\dot{w} = f_{OFF} \sinh\left(\frac{i}{i_{OFF}}\right) \exp\left(-\exp\left(\frac{w - a_{OFF}}{w_c} - \frac{|i|}{b}\right) - \frac{w}{w_c}\right) \tag{4}$$

Chapter 3: Implication Logic Based Circuits

3.1 Overview

Memristor logic described here is based entirely on the concept of logic implication, as described in [3]. An operation $Q^+ \leftarrow P$ imp Q equates to the logic operation $Q^+ = P' + Q$. The memristive states are set by limiting the "compliance current" by R_L [31]. The fan-out of memristors is limited to 1. For example, $Q^+ \leftarrow P$ imp Q and $R^+ \leftarrow P$ imp R cannot happen simultaneously when Q and R are not equal. There will be interactions between Q and R, which will not result in the expected logic operation.

In logic design, there will often be a requirement for high fan-out. Multiple copies of the input need to be made [4]. If Q and R = 0, then by changing R_L to account for appropriate compliance current values, then the two operations $Q^+ \leftarrow P$ imp Q and $R^+ \leftarrow P$ imp R can take place simultaneously as shown in Figure 23. This helps make multiple copies of P in just 2 cycles. In designs like the carry select adder, where the carry is required to drive many MUX blocks, this technique helps reduce the delay involved in making multiple copies of the input.

Previously [32], the concept of multi-fan out NAND (MFNAND) has been mentioned. But, the work fails to mention the circuit changes required to make the operation successful. Modification of the series resistor is essential to make the multi-fanout operation successful. The series resistor is used only to limit the current (compliance current, [31]) when the memristor changes its state from OFF to ON. The state change is happening only because sufficient voltage is developed across the memristor. The load resistor R_L reduces the power requirements of the excitation source (memristors in ON state has roughly 1k-2k ohm of resistance, and even with a 1V excitation source, the current in the branch will be 1mA, which is huge). Compliance current is the current limit which ensures that the memristor switches to a good-enough state to reliably perform logic

operations. When multiple memristors undergo an implication operation as shown in Figure 1, the current needed to switch them will also increase. This is the reason R_L is reduced when more than one memristor is involved in the implication operation.

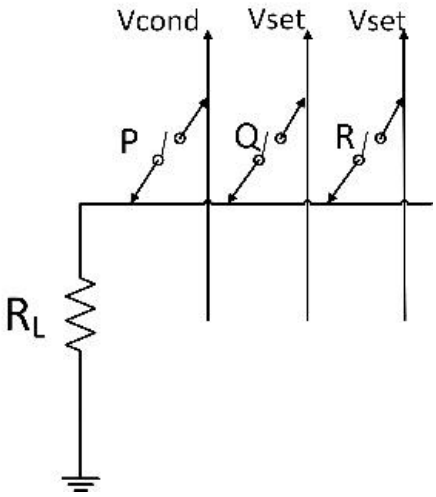


Figure 23: Multi fan out operation sequence.

3.2 DESIGN OF BASIC GATES

In this section, the designs of basic gates are explained. This provides the basis for the assumptions made with delay and area estimation in the design of the various adders in the paper. For the sake of generality, A and B are assumed as the input memristors. M1, M2, etc. are temporary memristors used to perform the necessary operations. All temporary memristors are assumed to be in their reset state (M1 = 0, M2 = 0 etc.), which can be done by applying V_{reset} at the beginning of the computation sequence. It is interesting to note that the input registers (A, B) are unaltered in this design. Only the temporary registers used are modified. Hence, the hardware numbers quoted are the number of temporary memristors used per gate. Some of the designs has been explained in [4], but comparatively, the design elucidated below has lower hardware count as well as time delay.

3.2.1 NAND Gate

Y = (AB)' = A' + B'

Y = A imp (B imp 0)

Sequence of operations:

Cycle 1: $M1^{+} \leftarrow$ B imp M1 ($M1^{+}$ = B imp 0 = B')

Cycle 2: $M1^{++} \leftarrow$ A imp $M1^{+}$ ($M1^{++}$ = A imp $M1^{+}$)

Delay = 2 cycles

Hardware = 1

3.2.2 AND Gate

Y = AB = (A' + B')'

Y = [A imp (B imp 0)] imp 0

Sequence of operations:

Cycle 1: $M1^{+} \leftarrow$ B imp M1 ($M1^{+}$ = B imp 0 = B')

Cycle 2: $M1^{++} \leftarrow$ A imp $M1^{+}$ ($M1^{++}$ = A imp $M1^{+}$)

Cycle 3: $M2^{+} \leftarrow M1^{++}$ imp M2 ($M2^{+} = M1^{++}$ imp 0)

Delay = 3 cycles

Hardware = 2

3.2.3 NOR Gate

Y = (A+B)'

Y = [(A imp 0) imp ((B imp 0) imp 0)] imp 0

Sequence of operations:

Cycle 1: $M1^{+} \leftarrow$ B imp M1 ($M1^{+}$ = B imp 0 = B')

$M2^{+} \leftarrow$ A imp M2 ($M2^{+}$ = A imp 0 = A')

Cycle 2: $M3^{+} \leftarrow$ M1+ imp M3 ($M3^{+} = M1^{+}$ imp 0 = B)

Cycle 3: $M3^{++} \leftarrow$ M2+ imp $M3^{+}$

Reset M1 (M1 = 0)

Cycle 4: $M1^{+} \leftarrow M3^{++}$ imp M1

Delay = 4 cycles

Hardware = 3 memristors.

3.2.4 OR Gate

Y = A+B

Y = (A imp 0) imp ((B imp 0) imp 0)

Sequence of operations:

Cycle 1: $M1^{+} \leftarrow$ B imp M1 ($M1^{+}$ = B imp 0 = B')
$M2^{+} \leftarrow$ A imp M2 ($M2^{+}$ = A imp 0 = A')
Cycle 2: $M3^{+} \leftarrow M1^{+}$ imp M3 ($M3^{+} = M1^{+}$ imp 0 = B)
Cycle 3: $M3^{++} \leftarrow M2^{+}$ imp $M3^{+}$

Delay = 3 cycles
Hardware = 3 memristors.

3.2.5 XOR Gate

Y = A'B+AB'
Y = (A+B') imp (AB')
Y = (B imp A) imp ((A imp B) imp 0)
Y = (B imp (A imp 0 imp 0)) imp ((A imp (B imp 0 imp 0)) imp 0)
Sequence of operations:
Cycle 1: $M1^{+} \leftarrow$ B imp M1 ($M1^{+}$ = B imp 0 = B')
$M2^{+} \leftarrow$ A imp M2 ($M2^{+}$ = A imp 0 = A')
Cycle 2: $M3^{+} \leftarrow M1^{+}$ imp M3 ($M3^{+} = M1^{+}$ imp 0 = B)
$M4^{+} \leftarrow M2^{+}$ imp M4 ($M4^{+} = M2^{+}$ imp 0 = A)
Cycle 3: $M3^{++} \leftarrow$ A imp $M3^{+}$
$M4^{++} \leftarrow$ B imp $M4^{+}$
Reset M1, M2
Cycle 4: $M1^{+} \leftarrow M3^{++}$ imp M1
Cycle 5: $M1^{++} \leftarrow M4^{++}$ imp $M1^{+}$

Delay = 5 cycles
Hardware = 4 memristors.

Table 1: Summary of the Characteristics of Basic Gates

Unit	Delay	Hardware
NOT	1	1
NAND	2	1
AND	3	2
NOR	4	3
OR	3	3
XOR	5	4

Memristor models are used to simulate and obtain the delay and energy numbers. The model for the memristor is exactly the same as in [27]. A VTEAM model [29] with Biolek window function is used with the following parameters. k_{on} = -216m/sec, k_{off} = 0.091m/sec, $V_{T,ON}$ = -1.5V, $V_{T,OFF}$ = 0.3V, x_{on} = 0, x_{off} = 3mm, a_{on} = 4, a_{off} = 4, R_{ON} = 1k ohm, R_{OFF} = 300k ohm. The transistors used are thick oxide devices from 180nm planar CMOS process. Better performance can be expected with shorter channel length transistors.

Memristor as Driver (MAD for short) gates [27] relies on the use of threshold switches. Design of a threshold switch is complicated. It requires either a novel device which can switch completely ON/OFF in the required voltage range, at which point, its compatibility with CMOS process is in question, or a comparator which triggers at the required voltage range to drive a switch. The switches need to turn ON completely to offer minimal resistance to current in the result branch. For example, in the realization of an OR gate, the threshold range for the switch to activate/turn ON is between 0.07V to 0.13V. This is a very narrow range for any switch to work. It demands the use of a high gain comparator to activate the switch, adding to additional overhead. Also, due to noise, process variations (offsets induced in the threshold switch, change in the threshold range due to changes in resistance of memristors/switches etc.) the idea may no longer be feasible. These issues have not been described in the literature.

A comparison of the delay/hardware required and energy is given below. Table 2 provides delay numbers for different designs which uses implication logic for logic operations. This delay does not include the set/reset/input initialization cycles like the other works. As explained before, the set/reset operations can all run in parallel at the time inputs are initialized. Table 3 compares the hardware required for the logic operations. Table 4 provides the energy consumption. The energy consumption numbers are not provided for

[4,5] implementations. Since energy numbers depend on the model of memristor used, applied voltage, compliance current limited resistors etc., it is hard to do a fair apple-apple comparison.

Table 2: Gate delay comparison

Unit	[4,5]	Current Work
NOT	1	1
AND	3	3
NAND	2	2
OR	4	3
NOR	5	4
XOR	8	5
Copy	2	2

Table 3: Hardware comparison (Memristor, Resistor)

Unit	[4,5]	Current Work
NOT	2, 1	2, 1
AND	4, 1	4, 1
NAND	3, 1	3, 1
OR	6, 2	5, 2
NOR	6, 2	5, 2
XOR	7, 3	6, 3
Copy	2, 1	2, 1

Table 4: Energy (Joules)

Unit	Current Work
NOT	2.7e-14
AND	8.1e-14
NAND	5.4e-14
OR	8.1e-14
NOR	10.8e-14
XOR	13.5e-14
Copy	5.4e-14

3.3 DESIGN OF A FULL ADDER

A full adder is shown in Figure 24. A, B, C_{in} are the inputs. S is the sum and C_{out} is the carry output. Let S_1, C_{i1} and C_{i2} be the intermediate sum and carries respectively. The main problem with memristors is the fan-out limitation of 1, unless it is in a copy operation as explained in Section 3.1. When designing adders, the fan-out limitation must be considered, because many intermediate signals (like S_1) drive multiple gates. Certain signal paths need to be prioritized over others to minimize the overall delay.

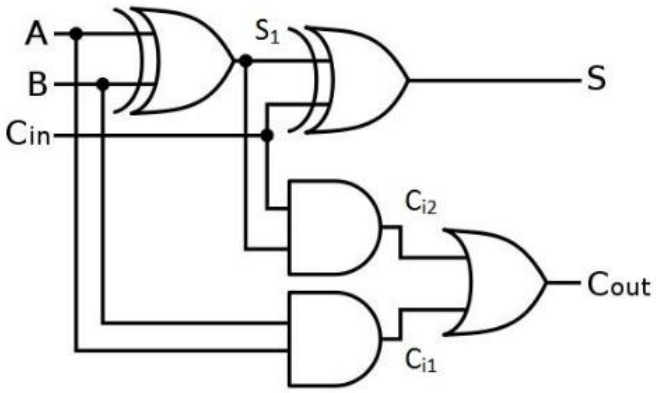


Figure 24: Full Adder Circuit.

The design is done as follows. To generate S_1 and C_{i1}, the inputs need to drive both the AND gate and XOR gate. From Section 3.2, it can be noted that in the first cycle, both AND and XOR requires A' or B'. After the first cycle, XOR would not need A and B till cycle 3. So, the cycle 2 of AND can use A and perform the operation parallel. Thus, S_1 takes the usual XOR delay which is 5 delays, C_{i1} takes 3 cycles. The goal of the design is to minimize the input to carry delay. It is assumed that A, B, and C_{in} arrive at the same time. Hence, (C_{in} imp 0) of Equation (4a) can be calculated before the arrival of S_1. C_{i2} is generated as follows:

$C_{i2} = S_1 C_{in}$
$C_{i2} = (S_1' + C_{in}')'$
$\boldsymbol{C_{i2} = (S_1 \text{ imp } (C_{in} \text{ imp } 0)) \text{ imp } 0}$ (4a)

C_{i2} is an AND operation between S_1 and C_{in}. S_1 also feeds the XOR gate to calculate the final sum S. In the first stage of the full adder, the inputs generate A' and B' by doing (A imp 0) and (B imp 0), after which they are available for both the XOR and AND gates to run in parallel, but S_1 cannot do that in the first cycle of the second stage of the full adder. In order to speed up carry generation, the sum generation is delayed by 1 cycle. Once S_1 imp C_{in}' is completed, S_1 is released to the XOR gate to generate the final sum. The final carry C_{out} is an OR operation between C_{i1} and C_{i2}. The operation is shown below.

$C_{out} = C_{i1} + C_{i2}$
$C_{out} = C_{i2}'$ imp C_{i1}
$C_{out} = \boldsymbol{(C_{i2} \text{ imp } 0)}$ imp C_{i1} (4b)

The highlighted sections in Equations (4a) and (4b) are redundant. Instead of generating C_{i2}, C_{i2}' is generated and input to the OR gate to calculate the final carry [4]. By doing this, 2 implication cycles can be saved. From all the above arguments, the delays can be calculated as follows.

A,B -> S = 5 + 1 + 5 = 11
A,B -> C_{out} = 5 + 1 + 1 = 7
C_{in} -> C_{out} = 1 + 1 + 1 = 3
C_{in} -> S = 5

In the case of a ripple carry adder, the carry-carry delay becomes significant. In Equation (4a), instead of (C_{in} imp 0), (S_1 imp 0) can be performed, and kept ready for the arrival of C_{in}. Equation (4a) can be modified as follows. This reduces the carry-carry delay to 2 cycles.

$C_{i2}' = (S_1 C_{in})'$

$C_{i2}' = S_1' + C_{in}'$

$C_{i2}' = (C_{in} \text{ imp } (S_1 \text{ imp } 0))$ (4c)

So, the delay, from C_{in} to C_{out} = 2 cycles. It is important to note that this design is adopted only in the intermediate stages of a ripple carry adder. The first and the last stage will still follow the design specified by Equation (4a).

The hardware requirement for a full adder is as follows. The 2 XOR gates consume 4 memristors each. The AND gate for C_{i1} takes 2 and the NAND gate for C_{i2} takes 1. The OR gate operates on intermediate signals, which need not be preserved. Hence, the OR gate doesn't consume any extra memristors. So, a total of 11 memristors per full-adder is required.

3.4 RIPPLE CARRY ADDER

The ripple carry adder (RCA) [33] is a cascade of full adders. This section shows the delay/area requirements for a ripple carry adder designed with the full adders explained in the previous section.

Consider an N-bit ripple carry adder, with inputs (A_0, B_0) to (A_{N-1}, B_{N-1}) and a carry input C_{in}. C_1, C_2 to C_{N-1} are all intermediate carry signals. C_N is the final carry-out. The sum bits are S_0 to S_{N-1}. The hardware required is 11N. The delay for an N-bit RCA is calculated as follows.

A_0, B_0 -> C_1= 7 cycles
# of intermediate stages = N-2
Total carry-carry delay = 2 x (N-2) = 2N – 4
Final stage (C_{N-1} – S_{N-1}) delay = 5
(C_{N-1} – C_N) delay = 3

Total delay to sum = **2N + 8**
Total delay to carry = **2N + 6**

Table 5: Delay to Sum comparison

Bits	Delay		
	[4]	[18]	Current
N	8N+12	5N+18	2N+8
16	140	98	40
64	524	338	136
256	2060	1298	520
1024	8204	5138	2056

3.5 CARRY SELECT ADDER

The carry select adder (CSA) [33] uses RCAs, 2:1 multiplexers and carry logic. It divides the words to be added into blocks and forms two sums for each block in parallel (one with a carry in of ZERO and the other with a carry in of ONE). As shown for a 16-bit carry select adder in Figure 25, the carry out from the previous block controls a multiplexer that selects the appropriate sum. The carry out is computed using the carry out of the adder with a carry input of ONE (shown as $p_{j:i}$ on the figure) and the carry out of the adder with a carry input of ZERO (shown as $g_{j:i}$ on the figure).

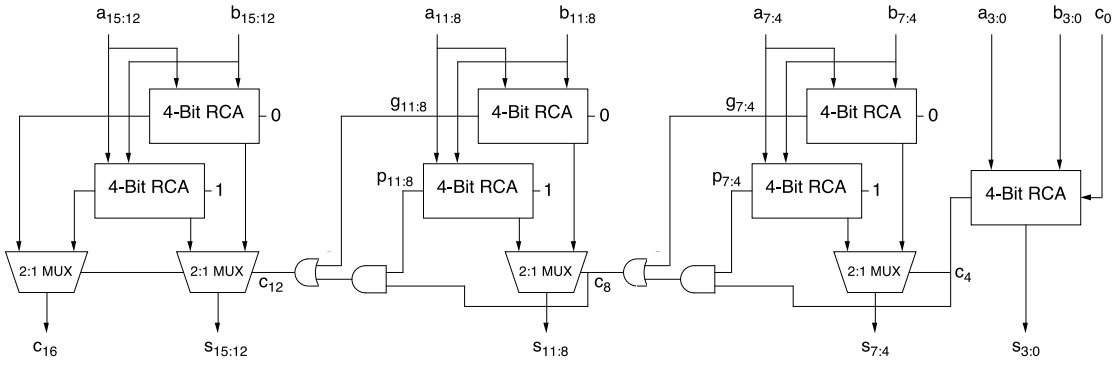


Figure 25: 16-bit Carry Select Adder.

3.5.1 RCA design

In the RCA design explained in Section 3.4, the input to sum delay is optimized by using full adders specified by Equation (4a) for the first and last stage, and the full adders specified by Equation (4c) for the intermediate stages. For a CSA, the delay from the inputs to the carry output becomes more important. In such designs, by having full adders specified by Equation (4a) for the first stage and that by Equation (4c) for the rest of the stages reduces the input to carry delay by 1 cycle, but increases the input to sum delay by 1 cycle. So, the RCA design for a carry select adder uses the above mentioned technique where the input to carry delay is minimized. Thus,

Total delay to sum = 2N + 9
Total delay to carry = 2N + 5

3.5.2 Multiplexer design

A multiplexer with inputs A and B and select line S can be represented by the equation, Y = S'A + SB. In a CSA, the carry output of the previous stage selects the appropriate sum. It is important to reduce this carry to sum delay. Hence, the implementation with memristors will be as follows. Analyzing the equation, it is clear that the delay with respect to the select line S is 4. Also, the hardware required is 4 memristors.

Y = S'A + SB
Y = (B' + S')' + (S + A')'
Y = [B imp (S imp 0)] imp [((S imp 0) imp (A imp 0)) imp 0]

3.5.3 Carry logic

For a simple 16-bit adder, with an RCA block size of 4 bits, the carry equations are:

$C_8 = g_{7:4} + p_{7:4} C_4$
$C_{12} = g_{11:8} + p_{11:8} C_8$

Each is realized with a single memristor and a delay of 2 cycles.

$C_8 = [C_4 \text{ imp } (p_{7:4} \text{ imp } 0)] \text{ imp } g_{7:4}$
$C_{12} = [C_8 \text{ imp } (p_{11:8} \text{ imp } 0)] \text{ imp } g_{11:8}$

It is important to note that C_8 and C_{12} not only drive the carry logic block, but also the multiplexers that select the appropriate sum. The two operations cannot be performed in parallel. Hence, the carry generation is prioritized for the intermediate stages and sum selection for the last stage. The delay for a N bit CSA, with a RCA block size of k is calculated as follows. In the calculation of the delay in carry logic for the third stage, it was assumed that ($p_{11:8}$ imp 0) would be ready by the time C_8 is ready. However, for the second stage, $p_{7:4}$ arrives at the same time as C_4, hence increases the delay by 1 cycle. This happens only with the 2nd stage. The lowest delay is achieved when $\mathbf{k = \sqrt{N}}$.

1st stage RCA input to carry delay = 2k+5
# of intermediate stages = N/k – 2
Carry-carry delay = [2 x (N/k – 2)] + 1= 2N/k – 3
Last stage carry to sum MUX delay = 4
Total delay to sum = **2N/k + 2k + 6**
Total delay to carry = **2N/k + 2k + 5**

The hardware requirement is as follows.

RCA = 11k x [1 + 2 x (N/k – 1)] = 22N – 11k
MUX = 4k x (N/k – 1) = 4N – 4k
Carry Logic = N/k – 1
Total hardware = **26N – 15k + N/k – 1**

3.6 CARRY LOOK-AHEAD ADDER

A Carry look-ahead adder (CLA) is shown in Figure 26. A CLA has 2 main blocks. The modified full adder stage and the logic stage. The operation of the modified full adder (MFA) stage is explained here. The MFA does both the SUM calculation and g_k/p_k generation. The carry to sum delay is 5 cycles in a full adder. The sum generation consumes

8 memristors (2 XOR gates in series). The description of the look-ahead logic is described in Sections 3.7 and 3.8.

$g_k = a_k b_k = \{a_k \text{ imp } (b_k \text{ imp } 0)\} \text{ imp } 0$
$p_k = a_k + b_k = (a_k \text{ imp } 0) \text{ imp } b_k$
Total delay = **3 cycles**
Hardware = 3 + 8 = **11 memristors**

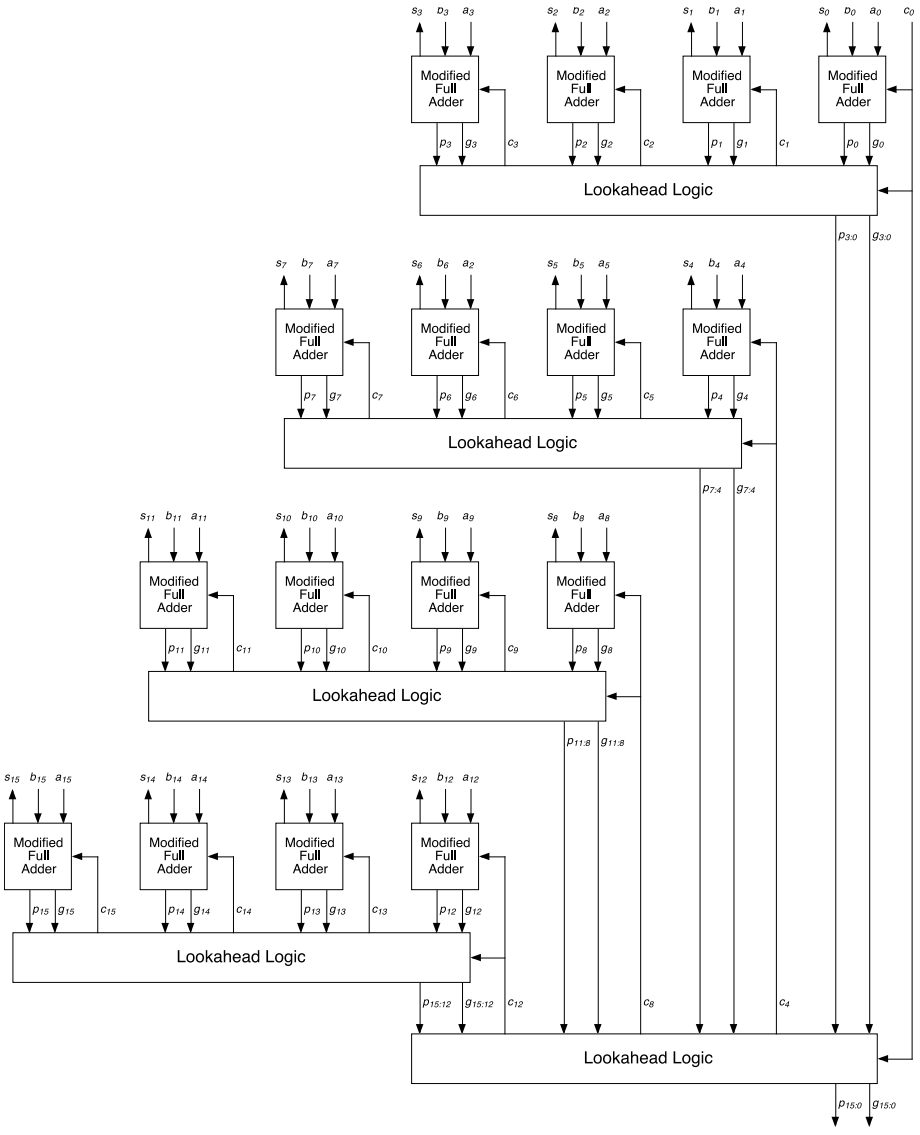


Figure 26: 16 bit Carry Look-ahead adder.

3.7 RADIX-2 CARRY LOOK-AHEAD ADDER

In this section, the design of a logic block of size 2 bits is described. The computations in the logic block are as follows.

$g_{k:k-1} = g_k + p_k g_{k-1}$
$p_{k:k-1} = p_k p_{k-1}$
$C_k = g_{k-1} + p_{k-1} C_{k-1}$

The design of the first level logic block is different from other levels. For example, the first level equations are as follows.

$G_{1:0} = g_1 + p_1 g_0 = (p_1 \text{ imp } g_0') \text{ imp } g_1$
$P_{1:0} = p_1 p_0 = (p_1 \text{ imp } (p_0 \text{ imp } 0)) \text{ imp } 0$
$C_1 = g_0 + p_0 C_0 = (C_0 \text{ imp } (p_0 \text{ imp } 0)) \text{ imp } g_0$

It should be noted that g_k' is obtained directly from the first stage (MFA). This helps speed up some calculations by avoiding concurrency issues. The sequence of operations is as follows, and is strictly w.r.t the first level of the logic block.

Cycle 1: $(p_1 \text{ imp } g_0')$
2x counts of $(p_0 \text{ imp } 0)$
Cycle 2: $G_{1:0} = (p_1 \text{ imp } g_0') \text{ imp } g_1$
$P_{1:0}' = p_1 p_0 = (p_1 \text{ imp } (p_0 \text{ imp } 0))$
$(C_0 \text{ imp } (p_0 \text{ imp } 0))$
Cycle 3: $P_{1:0} = p_1 p_0 = (p_1 \text{ imp } (p_0 \text{ imp } 0)) \text{ imp } 0$
$C_1 = g_0 + p_0 C_0 = (C_0 \text{ imp } (p_0 \text{ imp } 0)) \text{ imp } g_0$

So, $G_{1:0}$, $P_{1:0}$' is ready in 2 cycles, $P_{1:0}$ and C_1 has a delay of 3 cycles in the first stage. For the second level, the design changes slightly. G_k' is no longer generated for free from level 1, instead P_k' is available. So, the sequence of computation is changed for the other levels as shown below. It should be noted that G_1/P_1 after the first level represents $G_{3:2}/P_{3:2}$ and G_0/P_0 represents $G_{1:0}$ and $P_{1:0}$ and so on. For the logic blocks which receives the carry directly from C_0, C1 can be calculated with a 2 cycle delay as C_0' can be

precomputed. But, the logic blocks on the critical path which gets its input from $C_8/C_{12}/C_{14}$ etc., $C_1 = G_0 + P_0C_0 = (P_0 \text{ imp } C_0') \text{ imp } G_0$ will be changed to $C_1 = G_0 + P_0C_0 = (C_0 \text{ imp } P_0') \text{ imp } G_0$ as P_0' can be precomputed. This helps save 1 cycle in the carry generation path.

$G_{1:0} = G_1 + P_1G_0 = (G_0 \text{ imp } P_1') \text{ imp } G_1$
$P_{1:0} = P_1P_0 = (P_1 \text{ imp } P_0') \text{ imp } 0$
$C_1 = G_0 + P_0C_0 = (P_0 \text{ imp } C_0') \text{ imp } G_0$
Cycle 1: $(G_0 \text{ imp } P_1')$
$(P_1 \text{ imp } P_0')$
$(P_0 \text{ imp } C_0')$
Cycle 2: $G_{1:0} = (G_0 \text{ imp } P_1') \text{ imp } G_1$
$P_{1:0} = (P_1 \text{ imp } P_0') \text{ imp } 0$
$C_1 = (P_0 \text{ imp } C_0') \text{ imp } G_0$

For a N bit adder (where $N = 2^k$), the delay to the sum is

Pre-processing stage delay = 3
P/G -> P/G delay = 2 x #levels = 2 x (k-1)
P/G -> C delay in the last level = 2
C->C delay = 2 x #levels = 2 x (k-1)
C_{15} -> S_{15} delay = 5
Total delay to sum= 4k+6
Total delay to carry = 2k+5

3.8 RADIX-4 CARRY LOOK-AHEAD ADDER

In this section, the design of a logic block of size 4 bits is described. The computations in the logic block are as follows. The design for a 4-bit logic block is a bit more complicated, owing to the fact that the fan out is very limited in memristors, which adds to the latency. The equations for the 1st level block are as follows.

$C_1 = g_0 + p_0C_0$
$C_2 = g_1 + p_1g_0 + p_1p_0C_0$
$C_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_0$
$g_{3:0} = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$

$p_{3:0} = p_3p_2p_1p_0$

Multiple copies of g/p/c are needed to proceed with the logic operations. This will add an overhead of 2 cycles per level of logic. Since there are multiple copies, there is no need to be concerned about concurrency issues. Each evaluation can be treated independently.

All numbers reported in the following slides include the 2 cycle delay involved in the copy operation. $C_1/C_2/C_3/g_{3:0}/p_{3:0}$ takes 3/5/7/7/6 cycles to evaluate respectively. The evaluation of $g_{3:0}$ and $p_{3:0}$ is similar to C_3.

$C_1 = g_0 + p_0C_0 = (C_0 \text{ imp } p_0') \text{ imp } g_0$
$C_2 = g_1 + p_1g_0 + p_1p_0C_0 = \{p_1 \text{ imp } (C_0 \text{ imp } p_0')\} \text{ imp } \{(p_1 \text{ imp } g_0') \text{ imp } g_1)$
$C_3 = (g_2 + p_2g_1) + (p_2p_1g_0 + p_2p_1p_0C_0)$
$C_{3a} = (g_2 + p_2g_1)' = \{(p_2 \text{ imp } g_1') \text{ imp } g_2\} \text{ imp } 0$
$C_{3b} = p_2p_1g_0 = (p_2 \text{ imp } (p_1 \text{ imp } g_0')) \text{ imp } 0$
$C_{3c} = (p_2p_1p_0C_0)' = \{(p_2 \text{ imp } p_1') \text{ imp } 0\} \text{imp}(p_0 \text{ imp } C_0')$
$C_3 = C_{3a} \text{ imp } (C_{3c} \text{ imp } C_{3b})$

To generalize, for a N bit adder (where N = 4k), we can find the delay to the sum as follows.

Pre-processing stage delay = 3
p/g -> p/g delay = 7 x #levels = 7 x (k-1)
P/G -> C delay in the last level = 7
C->C delay = 5 x #levels = 5 x (k-1)
C_N -> S_N delay = 5
Total delay to sum= 12k+3
Total delay to carry = 7k+6

Table 6 summarizes the difference in delays between CLA-2 and CLA-4. In the table $k_2 = \log_2 N$ and $k_4 = \log_4 N$. Thus, $k_2 = 2k_4$.

Table 6: Delay comparison of CLA Architectures

Bits	RCA	CLA-2	CLA-4
N	2N + 8	$4k_2 + 6$	$12k_4 + 3$
16	40	22	27
64	136	30	39
256	520	38	51
1024	2056	46	63

Table 7: Hardware comparison of CLA Architectures

Bits	RCA	CLA-2	CLA-4
16	176	237	351
64	704	957	1439
256	2816	3837	5791
1024	11264	15357	23199

3.9 CARRY SKIP ADDER

The carry skip adder is a modification of the CSA, with lesser hardware requirement and more delay. It divides the words to be added into blocks (like the carry select adder). The basic structure of a 16-bit carry skip adder implemented with five 3-bit blocks and one 1-bit-wide block is shown in Figure 27. Within each block, a ripple carry adder is used to produce the sum bits and the carry (which is used as a $g_{j:i}$ signal). In addition, an AND gate is used to form the $p_{j:i}$ signal. These signals are combined using carry logic (like that of the CSA) to produce a fast carry signal.

The delay for an N bit carry skip adder, with a RCA block size of k is calculated as follows. The carry generation requires the calculation of the $g_{j:i}$ signals, whose computation will occur in parallel to the 1st RCA block. It should be noted that as the size of k increases, the calculation of the $p_{j:i}$ signals becomes more complex, requiring more hardware, which scales with k.

1^{st} stage RCA input to carry delay = 2k+5
# of intermediate stages = N/k – 2

Carry-carry delay = [2 x (N/k – 2)] = 2N/k – 4
Last stage carry to sum RCA delay = 2k + 3
Total delay to sum = **2N/k + 4k + 4**
Total delay to carry = **2N/k + 2k + 4**

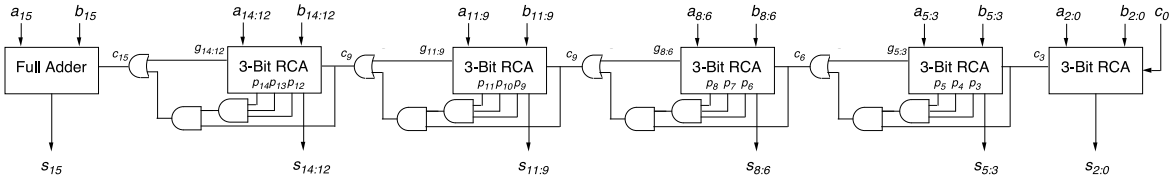


Figure 27: 16-bit Carry Skip Adder.

It can be shown that for minimum delay, the value of k should be k = √(N/2). The hardware requirement is as follows.

RCA = 11N
Carry Logic = k x (N/k – 1)
Total = **12N - k**

The following tables compare the delay and hardware requirements of different adders. The k value chosen is the optimum w.r.t delay minimization. So, k = √N for CSA and k = √(N/2) for carry skip adder.

Table 8: Delay to Sum comparison

Bits	Ripple Carry Adder	Carry Select Adder	Carry Skip Adder
N	2N + 8	2N/k + 2k + 6	2N/k + 4k + 4
8	24	18	20
16	40	22	28
32	72	30	36
64	136	38	52
128	264	50	68
256	520	70	92

Table 9: Area/Hardware comparison

Bits	Ripple Carry Adder	Carry Select Adder	Carry Skip Adder
N	11N	26N – 15k + N/k – 1	12N - k
8	88	165	94
16	176	359	189
32	352	747	380
64	704	1551	762
128	1408	3173	1528
256	2816	6431	3061

It clearly shows that a CSA has less delay than the other 2 architectures, but has a very big area penalty. A carry skip adder has reasonable delay and area requirements.

3.10 CONDITIONAL SUM ADDER

The conditional sum adder [33] consists of a full adder, modified half adders and multiplexers. A full adder realized with the above design methodology has an input to carry delay of 7 cycles. Modified half adders take 3 cycles to compute the carry signals, 5 cycles to compute the sum output. A multiplexer has a delay of 4 cycles from the select line to the output. The hardware requirements for a full adder, multiplexer and modified half adder is 11, 4 and 7 respectively. Taking these into consideration, the delay for a conditional sum adder is calculated as follows.

1st full adder inputs-carry delay = 7
# of intermediate MUX stages = $\log_2 N$
Delay per MUX stage = 4
Total delay to SUM/Carry = $\mathbf{7 + 4\log_2 N}$

Table 10: Metrics of Conditional Sum Adder

Bits	Delay	Hardware
2	11	26
4	15	54

8	19	130
16	23	318

3.11 PARALLEL PREFIX ADDER

Parallel prefix adders are very similar to a CLA. They improves the carry propagation by parallelizing the calculation of group generate and group propagate signals [7]. A typical adder consists of 3 main stages. For a n-bit adder, it is assumed that the inputs are A_k, B_k, C_1 where k = 1 to n.

3.11.1 Pre-processing stage

The first stage processes the input and calculate generate and propagate signals (g_k and p_k). The logic equations are as follows. It should be noted that the computation of g_k and p_k can occur simultaneously, because both gates have the first cycle computing (b_k imp 0) or (a_k imp 0).

$g_k = a_k b_k = \{a_k \text{ imp } (b_k \text{ imp } 0)\} \text{ imp } 0$
$p_k = b_k \text{ xor } a_k = (b_k \text{ imp } (a_k \text{ imp } 0 \text{ imp } 0)) \text{ imp } ((a_k \text{ imp } (b_k \text{ imp } 0 \text{ imp } 0)) \text{ imp } 0)$
Total delay = **5** cycles
Hardware requirement = **6** memristors

3.11.2 Prefix computation stage (P-G stage)

The prefix computation stage computes the group generate (G_k) and group propagate (P_k) signals for the carry calculation in the post-processing stage. There are different architectures which implement the P_k/G_k generation, but the fundamental equations remain the same across all designs. The first level equation is given below. It takes the g_k, p_k inputs from the pre-processing stage. It is important to note that p_k needs to un-altered as it is required by the post-processing stage for the final sum calculation.

$G_{i:k} = g_i + p_i g_k = (p_i \text{ imp } g_k') \text{ imp } g_i$
$P_{i:k} = p_i p_k = (p_i \text{ imp } (p_k \text{ imp } 0)) \text{ imp } 0$
Cycle 1: $p_i \text{ imp } g_k'$

p_k imp 0
Cycle 2: (p_i imp g_k') imp g_i
p_i imp p_k'
Cycle 3: (p_i imp p_k') imp 0

Most designs have a fan-out of just 2 in the first level. Since g_k' is readily available from the pre-processing stage, it can be used. Because of this, concurrency issues are avoided. The first level takes 3 cycles to compute, and 2 memristors.

For the second level, it should be noted that G_k' are no longer available readily, but P_k' are available. This is very similar to the design of a CLA. Hence, the equations must be modified as shown below. $G_{m-1,k}/P_{m-1,k}$ cannot be destroyed, as this is used in dot operator m-1 at the same time. To accommodate this, couple of changes are made for 2nd level and beyond. Also, the dot operator at m-1 will not run into concurrency issue. The operator at i would not be using $G_{m-1,k}/P_{m-1,k}$ when operator m-1 is using them.

$G_{i:k} = G_{i,m} + P_{i,m}G_{m-1,k} = (G_{m-1,k}$ imp $P_{i,m}')$ imp $G_{i,m}$
$P_{i:k} = P_{i,m}P_{m-1,k} = (P_{m-1,k}$ imp $(P_{i,m}$ imp $0))$ imp 0

The delay still remains at 3 cycles per level as long as the fan out is 2 like in the case of Kogge-Stone architectures, but increases to 4 cycles if the fan-out increases beyond two, where the first 2 cycles are used to generate the required P/P'/G and the other 2 are used to evaluate the equation without any concurrency issues. The hardware required is 2 memristors per level if the fan out is 2. The memristors across levels can be reused. It is interesting to note that P_k frees up 2 memristors at the end of the calculation of P_k/G_k. These can be reset and made use of at later levels. To simplify and generalize, the total hardware required by the P-G stage would be 2 + 1.5x the maximum hardware a level would require per bit. For example, in a Kogge-Stone adder where the fan-out is 2, the hardware needed for the P-G block is 2 + 1.5(2) = 5 memristors per bit.

3.11.3 Post-processing stage

This is the last stage in a parallel prefix adder, where the group generate and propagate signals are used to compute the carry signals and generate the final sum. The design goes as follows. In the calculation of C_k, (C_1 imp 0) will be readily available before $P_{k-1:0}$ and $G_{k-1:0}$ are generated, as it takes only 1 cycle. Thus C_k takes 2 cycles. It should be noted that S_k depends on the calculation of C_k and takes 5 cycles, which is the XOR delay.

$C_k = G_{k-1:1} + P_{k-1:1}C_1 = [P_{k-1:1}$ imp $(C_1$ imp $0)]$ imp $G_{k-1:1}$
$S_k = p_k$ xor C_k
Total delay = 2 + 5 = **7**
Hardware required = **5**

The pre-processing and post-processing stages add to a total delay of 12 cycles. The hardware requirement is 11 memristors per bit from these 2 stages.

3.12 PARALLEL PREFIX ARCHITECTURES

There are multiple modifications to the P-G stage of the adder, which are very well-known in the CMOS world [33-37]. The design of such architectures and the associated complexities with memristors is explained below.

3.12.1 Brent – Kung Adder

Pre/Post processing delay = 12
Prefix computation stages = $2\log_2 n - 2$
There are no issues with multiple fan-out except for 1 level. So, delay is 3 cycles per level and 4 cycles for the level with the fan-out issue.
Total delay for the prefix stage = $3(2\log_2 n - 2 - 1) + 4$
Total delay = $12 + 6\log_2 n - 5 = 7 + 6\log_2 n$

Hardware required per bit = 11 + 5

Additional 4 memristors are needed for the n/2th bit because of a higher fan-out.
Total hardware required = 16n + 4

3.12.2 Kogge – Stone Adder

Pre/Post processing delay = 12
Prefix computation stages = $\log_2 n$
There are no issues with multiple fan-out. So, delay is 3 cycles per level.
Total delay for the prefix stage = $3\log_2 n$
Total delay = $12 + 3\log_2 n$

Hardware required per bit = 11 + 5
Total hardware required = 16n

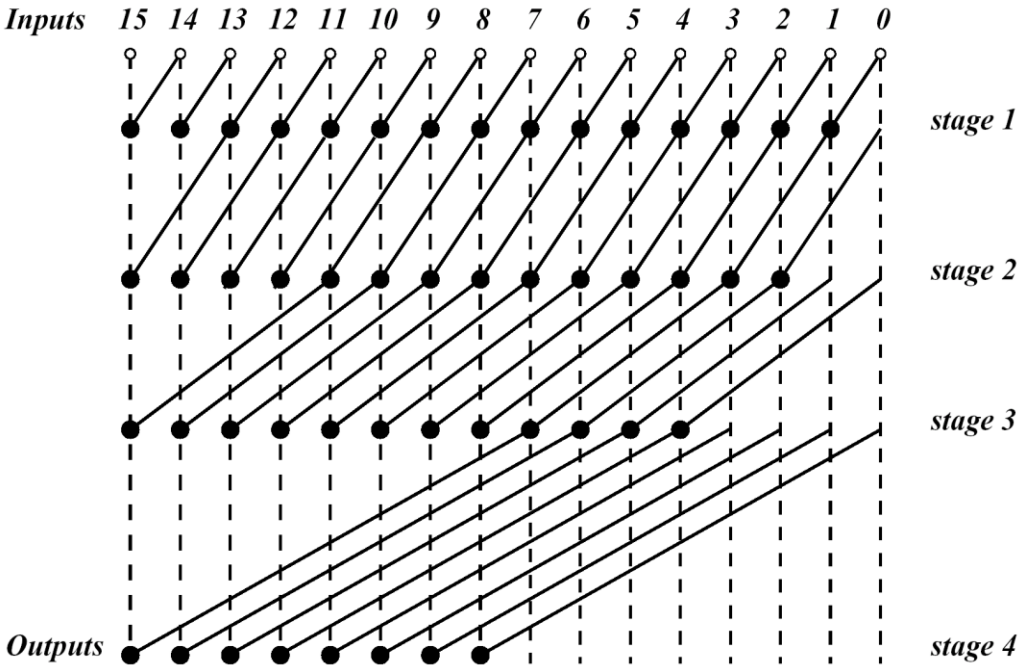


Figure 28: A typical 16 bit Kogge-Stone adder P-G stage.

3.12.3 Han – Carlson Adder

Pre/Post processing delay = 12
Prefix computation stages = $\log_2 n + 1$
There are no issues with multiple fan-out. So, delay is 3 cycles per level.
Total delay for the prefix stage = $3\log_2 n + 3$
Total delay = $15 + 3\log_2 n$

Hardware required per bit = 11 + 5
Total hardware required = 16n

3.12.4 Ladner – Fischer Adder

Pre/Post processing delay = 12
Prefix computation stages = $\log_2 n$
Except for the first level, the fan-out of every other stage is greater than 2. So, delay is 3 cycles for the first level and 4 cycles per level for all other levels.
Total delay for the prefix stage = $4(\log_2 n - 1) + 3$
Total delay = $12 + 4\log_2 n - 1 = 11 + 4\log_2 n$

The fan-out increases as a power of 2 in every stage, requiring multiple copies of P/G/P' to be made in order to keep the delay to the minimum.

Hardware required per bit = 11 + 5
Additional 1.5 x 4 x (n/2) memristors are needed for the $n/2^{th}$ bit because of a higher fan-out. Total hardware required = 19n

The following tables compare the delay and hardware requirements of different parallel prefix adder architectures. The comparison clearly shows that a Kogge-Stone architecture gives better delay performance among all the others due to the fact that its fan-out is limited to 2 at every level, and lesser levels compared to other architectures.

Table 11: Delay to Sum comparison

Bits(n)	Brent-Kung	Kogge-Stone	Han-Carlson	Ladner - Fischer
Stages	$2\log_2 n - 2$	$\log_2 n$	$\log_2 n + 1$	$\log_2 n$
Delay	$7 + 6(\log_2 n)$	$12 + 3(\log_2 n)$	$15 + 3(\log_2 n)$	$11 + 4(\log_2 n)$
4	19	18	21	19
8	25	21	24	23
16	31	24	27	27
32	37	27	30	31
64	43	30	33	35
128	49	33	36	39
256	55	36	39	43
512	61	39	42	47
1024	67	42	45	51

Table 12: Area/Hardware comparison

Bits(n)	Brent-Kung	Kogge-Stone	Han-Carlson	Ladner - Fischer
Stages	$2\log_2 n - 2$	$\log_2 n$	$\log_2 n + 1$	$\log_2 n$
HW	$16n + 4$	$16n$	$16n$	$19n$
4	68	64	64	76
8	132	128	128	152
16	260	256	256	304
32	516	512	512	608
64	1028	1024	1024	1216
128	2052	2048	2048	2432
256	4100	4096	4096	4864
512	8196	8192	8192	9728
1024	16388	16384	16384	19456

3.13 ARRAY MULTIPLIER

An array multiplier is shown in Figure 3. An N-bit array multiplier consists of N^2 AND gates, $N^2 - 2N$ full adders and N half adders. In an NxN array multiplier, it can be shown that that the critical path of the design consists of 1 AND gate delay, 2 half-adder

delays (input to sum and carry to carry delay), 2N-4 full adder delays (N-2 input to sum delay and N-1 carry to carry delay and 1 carry to sum delay).

Every input bit (a_i, b_i) drives N different AND gates. Due to concurrency issues with memristors, these operations cannot run in parallel. Hence, copies of the input are made in order to drive these AND gates. Using the technique described before, multiple copies of the input can be made in just 2 cycles. It can be shown that the first cycle of the AND gate and the second cycle of the copy operation can run in parallel. Hence, the effective delay of the AND gate (including the delay due to copy operations) is 4 cycles.

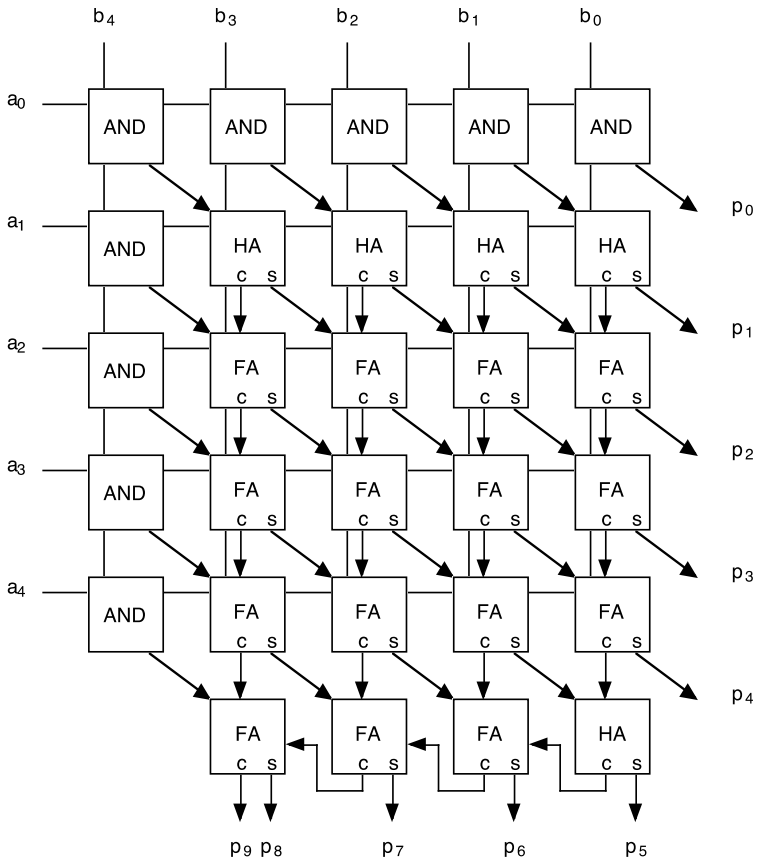


Figure 29: 5x5 array multiplier.

The half adder is designed as shown earlier. Hence, the input to sum delay is 5 cycles, with the carry to carry delay being 2 cycles. The full adder design changes along the critical path. The full adder is described earlier is optimized to have the minimum input to carry delay. The computation of S and C_{i2} cannot happen simultaneously (Figure 29). Hence, S_1 drives the logic to compute C_{i2} instead of S giving an input to sum delay increased of 11 cycles. But, for N-2 full adders in the array multiplier, S_1 will drive the logic to calculate S, instead of C_{i2}. The input to sum delay will be 10 cycles. N-1 of the full adders will be designed as described before, where the carry to carry delay is minimum (2 cycles). The last full adder has a carry to sum delay of 5 cycles. With this, the total delay to the MSB S_N can be shown as follows.

AND gate + copy cycle delay = 4 cycles
# of half adders = 2
Half adder delay = 5 + 2 = 7 cycles
# of full adders = 2N-4
Full adder delay = (N-2)x10 + (N-1)x2 + 5
Total input to MSB delay = **12N - 6**

The hardware requirement for an array multiplier is as follows. Each AND gate needs 3 memristors, full adders need 11 memristors, half adders need 6 memristors. Since the execution is sequential, it is possible to reuse the memristors used in previous cycles. For an NxN array multiplier, there are N-1 stages of full adders. Every stage has N-1 full adders and requires 11(N-1) memristors. It can be shown that all the other stages can be implemented with 2x the memristors needed per stage. Once the computation is completed in an ith stage, the memristors can be reset and used for $i+2^{th}$ stage. This reuse significantly saves the required hardware.

# of AND gates = N^2
Hardware for AND gates = $3N^2$
# of half adders = N
Hardware for half adders = 6N
# of full adders = N^2-2N
Hardware for full adders = 2 x 11(N-1)
Total hardware = $\mathbf{3N^2 + 28N - 2}$

Table 13: Metrics of an Array multiplier implemented with imply logic

Bits	Array Multiplier	
N	Delay	Hardware
5	54	213
8	90	414
16	186	1214
32	378	3966
64	762	14078

3.14 SUMMARY

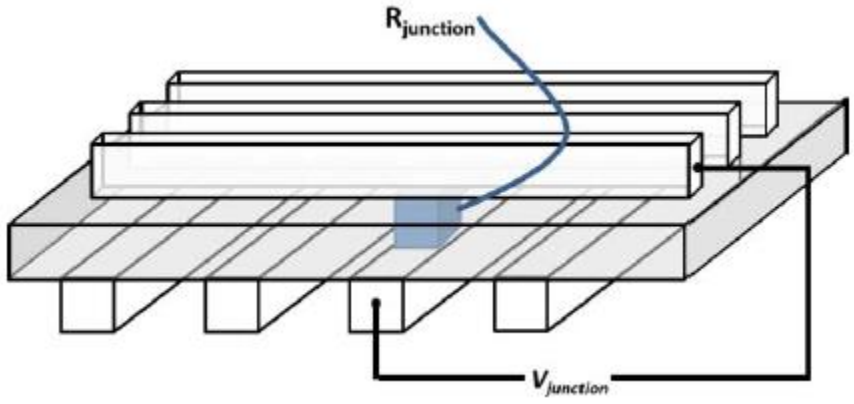
In this chapter, we have examined the delay and complexity of memristor implementations of conventional adders. It has been shown that the explained designs has better delay/power/area numbers compared to the state of the art. Simulation results of a few basic gates have been shown using the VTEAM model for memristors. An interesting result is that the carry look-ahead adder implemented with 2-bit look-ahead blocks is nearly as fast as and lower in complexity than the best of the parallel prefix adders, mainly due to the fan-out limitations in implication logic. The primary contributions of this chapter are as follows.

- It introduces a new sequence of imply-instructions to reduce the delay compared to the state of the art.
- This translates to significant reduction in the complexity of the control logic, area and power.

- We achieve this by making use of parallelism. Many operations are run in parallel which significantly helps speed up the computation of XOR, OR etc.
- Some redundant operations were identified and removed (for example, in full adders), thereby achieving 3x better delay numbers.
- Since memristor logic is sequential, most of the intermediate memristors are reused, thereby reducing the overall hardware count.

3.15 HIGH LEVEL PERSPECTIVE

Memristors are a usually a part of the cross-bar memory network. If the circuit were to be made reconfigurable, it is very difficult to control when a load resistor is connected to the required memristor etc. In a crossbar matrix, this will result in sneak-paths where the memristors can be falsely triggered, causing a change in state [26].



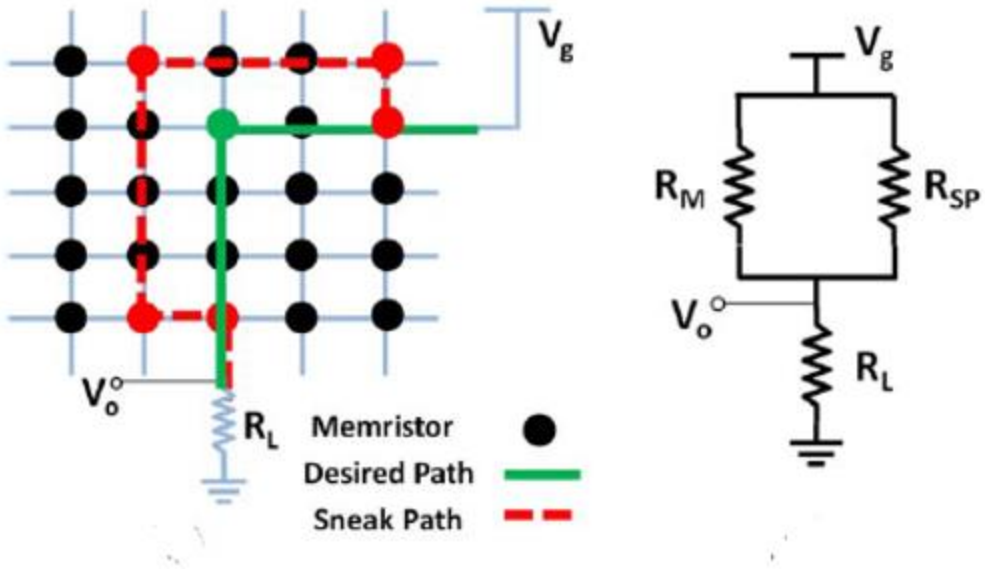


Figure 30: Sneak paths in crossbar architectures.

When a voltage is applied to a junction in the crossbar, current flows through paths different than the desired path (sneak paths). 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 Figure 30. This parallel resistance depends upon the stored data in the memristors. So, when sensing the state of the memristor, the parallel resistance affects the read voltage. It can also trigger false logic operations depending on the data in the memory.

To avoid this, it is important to move the computational memristors (also called the temporary memristors) out of the cross-bar matrix and treat them discretely like every other circuit element. Only the input and final output memristors will be in the memory crossbar matrix. All other temporary memristors will be a part of the logic circuit. This way, the design can be tremendously simplified.

Chapter 4: High Fan-out Memristor Logic

Implication logic has a major disadvantage when it comes to driving multiple gates. Since it is a current-mode operation, it is impossible to drive multiple gates with a single memristor at the same time as current sharing changes the logic functionality. This becomes a serious disadvantage for larger circuits like adders and multipliers as the complexity keeps increasing. Also, due to the large number of stages, the sequencing logic becomes more complicated. This chapter describes a new technique of designing memristor based logic gates by moving to voltage domain. Analog current mirrors can be used to sense currents in the input memristors and control the output memristor accordingly.

4.1 Current Mirrors

A current mirror is a circuit designed to copy a current from one branch to another. In network theoretical terms, it can be treated as a "Current Controlled Current Source" or CCCS. An ideal CCCS is expected to have 0 input impedance and infinite output impedance. Consider the current mirror shown in Figure 31. Here, a NMOS transistor, T_1 has its drain and gate terminals shorted. Such a connection is called "diode connection". When a current I_1 is forced through a diode connected transistor T_1, the gate voltage V_{GS1} it develops depends on the sizing of the device and the current according to the following equation.

$$I_d = k\,(V_{GS}-V_{th})^2$$

k is a constant dependent on the mobility, gate oxide thickness, doping concentration, width and length of the transistor. MOS transistors exhibit a square law behavior where the drain current is proportional to the square of the gate-to-source voltage,

V_{GS}. Now, when the same V_{GS1} is applied to another NMOS transistor T_2, the drain current in T_2 will be a multiple of I_1 and the ratio of the device sizes of T_2 and T_1. So, if T_2 is 2x larger than T_1, the drain current in T_2 is $2I_1$.

Current mirrors are trans-linear devices. They have a linear relation between the input and output current. This is true as long as the devices used are of the same type, with similar threshold voltages. It also assumes that the devices are in saturation to reduce the dependency on V_{DS}. Current mirror operation is valid across all regions of operation of a transistor, as long as the V_{DS} and V_{GS} are the same across the mirror pair. Changes in V_{DS} will result in change in the output current. There are many non-linearities that exist in CMOS current mirrors, especially in short channel technologies where the current ratio depends heavily on the process parameters and the drain-source voltage, but these are critical for analog applications. As far as digital logic goes, there is much room to play. More details about current mirrors and their working can be found in [38].

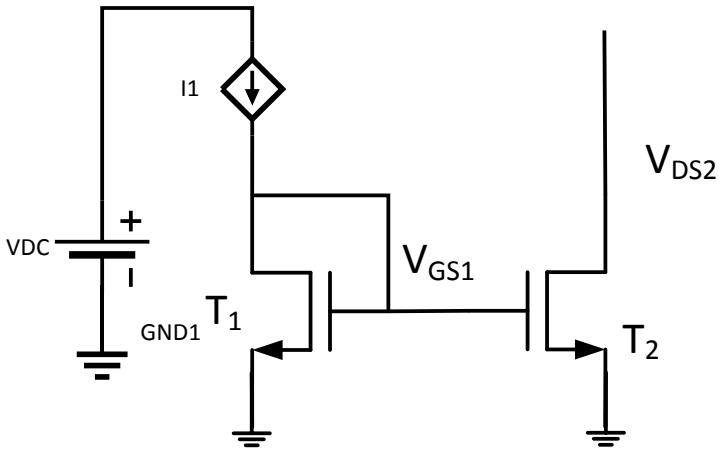


Figure 31: Basic current mirror operation

4.2 DESIGN OF BASIC GATES

In this section, the designs of basic gates are explained. For the sake of generality, M_A and M_B are assumed as the input memristors. From the previous chapter, it is clear that for a memristor to switch, it requires sufficient voltage (V_{SET}/V_{RESET}) and appropriate compliance current. The motivation to use current mirrors is as follows. If an input memristor is set, a current equal to the compliance current flows through it. This current is sufficient to flip the state of an output memristor (from reset to set state). If the input memristor is reset, the current is not enough to change the state of an output memristor.

By using current mirrors, the current in the input memristors is mirrored to the output memristor, thereby controlling the logic operation. The memristor used in this design [45] is a symmetric bipolar memristor. V_{COND} is 1.5V, V_{SET} is 3V, V_{RESET} is -3V. Logic operations with memristors happen in 2 stages - initializing the inputs and evaluating the outputs. A diode connected NMOS transistor requires at least (V_{th} + V_{ov}) dropped across it. V_{SET} is chosen such that it can switch the memristor with the given mirror load within 0.5ns.

The input memristors are either set or reset depending on the input value. The voltage sources V_1 and V_2 either drive V_{SET} or V_{RESET} depending on the value of A and B. In the same cycle, the output memristors need to be set/reset. For AND/OR/XOR/copy gates, the output memristor is reset by applying a V_{RESET} (V_{RESET} = -3V). For the inversion gates like NOT/NAND/NOR/XNOR, it can be seen from the figures below that they are flipped. So, applying V_{RESET} to the N terminal actually sets the memristor (changes it to a low resistance state).

The NMOS transistors T_A, T_B etc. in Figure 32 are sized such that when they are diode connected as shown, the resistance they offer is in the range of 5k – 10k. This is sufficient to act as a compliance current limiting resistor. Hence, there is no need to use an

external resistor to limit the current. In the next cycle, V_1 and V_2 take the values of V_{COND} (which in this case is 1.5V). If the memristors are set, it carries a current, which gets mirrored to the output memristors setting/resetting their states depending on the connections.

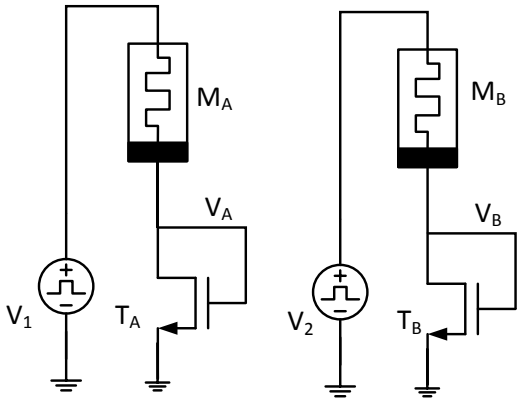


Figure 32: Input gates

To understand the circuit behavior, consider the implementation of an OR/NOR gate shown in Figure 33.

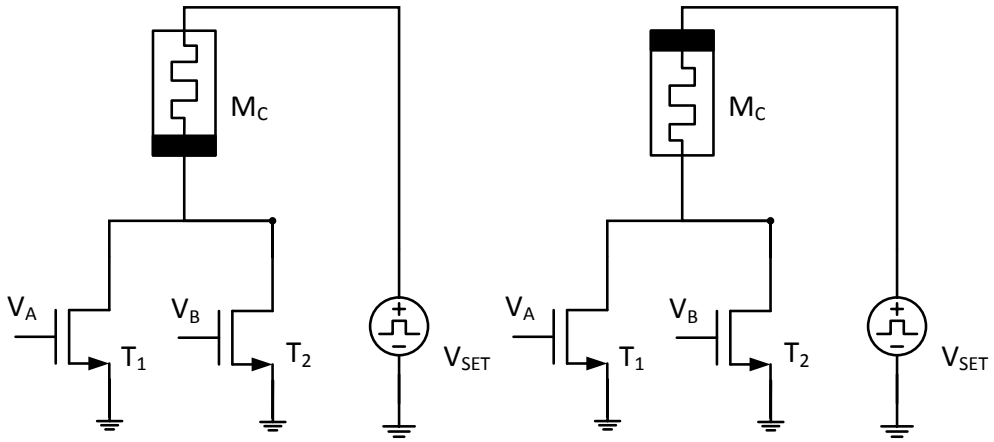


Figure 33: OR and NOR gates

The input memristors M_A and M_B are set/reset by V_1 and V_2. In the same cycle, M_C of the OR gate is reset. In the next cycle, V_1 and V_2 will be V_{COND} and V_{SET} will be applied on M_C. Consider a case where A = 1 and B = 0. When A = 1, transistor T_A has half the compliance current flowing through it (half because the applied voltage V_{COND} is half of V_{SET}). The transistor T_1 and T_2 are sized such that it has a width 2x that of T_A and T_B to replicate the required compliance current. Initially, the drain to source voltage of T_1 and T_2 will be very low because the transistors will be in linear region, creating a large voltage drop across the memristor and forcing it to switch to a low resistance state. Since M_C has V_{SET} applied to its terminals, it switches its state.

Eventually, the transistors move to saturation region and limit the current to the compliance current. M_C changes its state from reset to set (0 to 1), resulting in an OR operation. If A=B=0, then the currents through T_A and T_B are nowhere close to the compliance current. So, when V_{SET} is applied on M_C, a large voltage drop appears across the transistors and not across the memristor. Thus, the state does not change from reset to set. The principle of operation is very similar to a NOR gate. In this case, the memristor switches to a high resistance state when any one of the transistors are conducting.

The design of AND and NAND gates is shown in Figure 34. The working principle is similar to an OR gate design. T_2 is sized 2x of T_A for reasons stated above. T_1 is sized 6x of T_A. Assume a case where both M_A and M_B are set. The V_{GS} of T_A and T_B will be the same since they carry the same current. Since T_1 is stacked above T_2, the V_{GS} of T_1 will be less than the V_{GS} of T_2. To support the same current, the size needs to be very large. Hence it is sized 6x – 8x compared to T_A. Sizing it any larger will not help because once T_2 enters saturation, the current in the branch is limited by T_2. Any increase in the size of T_1 will only increase the V_{DS} of T_2 and not the current in the branch.

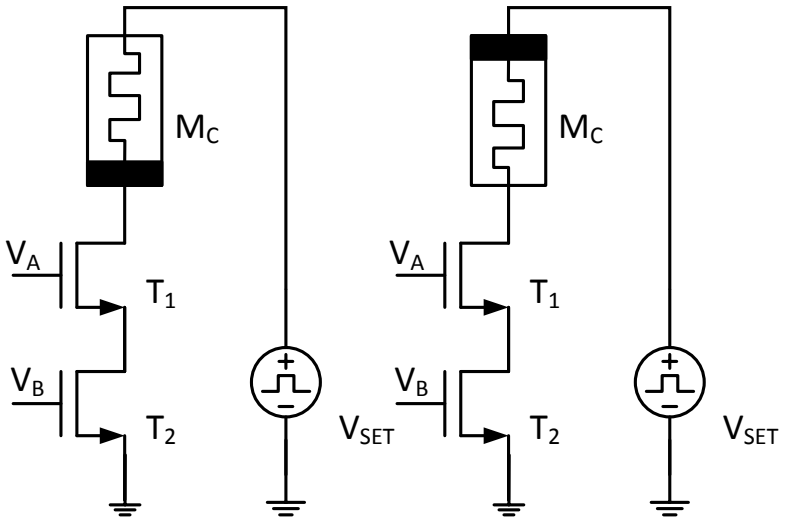


Figure 34: AND and NAND gates

The design of an XOR gate is shown in Figure 35. The sizing of the devices is important. T_1, T_{P1}, T_2, T_{P2} are sized similar to T_A and T_B. T_{P3}, T_{P4}, T_3, T_4 are 2x, with T_5 being 12x – 16x and T_6 at 4x (a rationale which follows AND gate design). The idea is as follows. For an XOR operation, C = A XOR B, C = B when A = 0; C = B' when A = 1. Designing for the case when A = 0, the operation is just a copy operation. It can be seen that M_C will follow B because when A = 0, T_3, T_5 is turned OFF. When A=1, if it wasn't for the branch T_5/T_6, M_C would be set irrespective of the state of B because there is at least 1 active path. The branch T_5/T_6 ensures that the current delivered by the PMOS transistors T_{P3} and T_{P4} doesn't reach the memristor only when A=B=1. The voltage across the memristor will be pulled low enough (less than the sum of V_{DS} of T_5 and T_6). This voltage will not be enough to switch the state of the memristor. Hence, when A=B=1, M_C remains in the reset state.

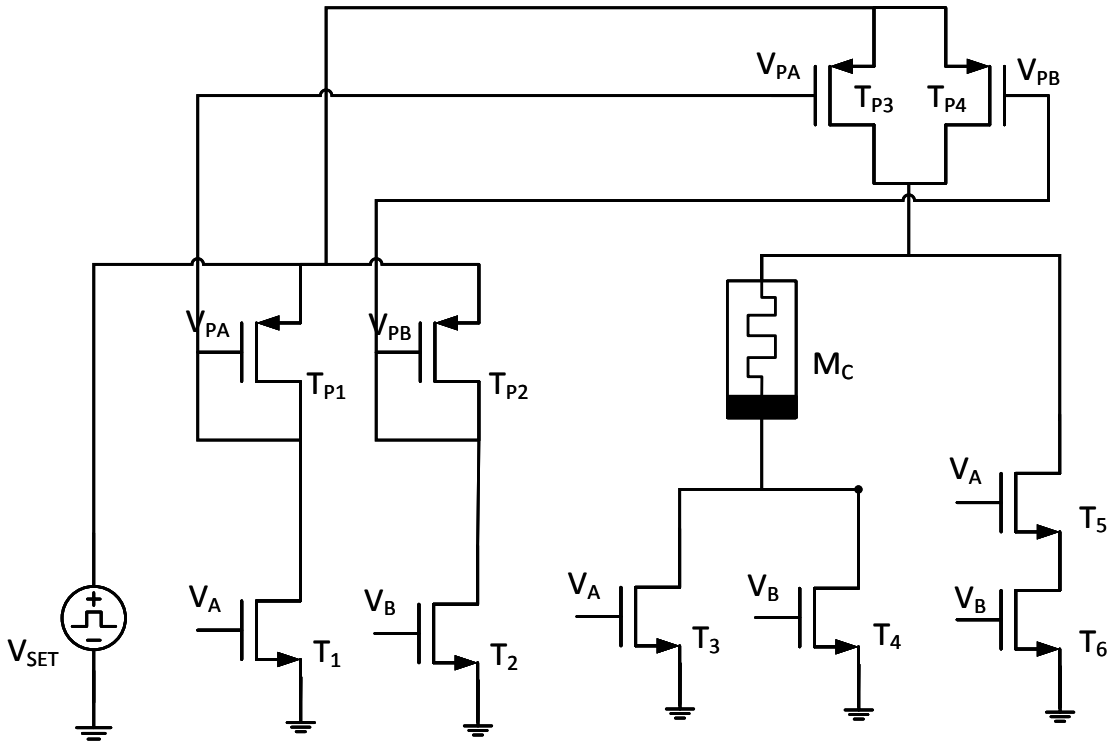


Figure 35: XOR gate

Using implication to realize logic gates as described in [4, 5] has many problems. The operation has limited fan-out. This limits the number of parallel operation that can run in a given instance. Also, the overhead associate with clock generation is enormous due to increased number of stages. MAGIC gates described in [26,39] suffer from similar concurrency issues. Also, the work doesn't describe the additional circuitry required to run the logic operations.

There are multiple versions of Memristor as Driver (MAD, for short) gates. One type [27] relies on the use of threshold switches. Design of a threshold switch is complicated. It requires either a novel device which can switch completely ON/OFF in the required voltage range, at which point, its compatibility with CMOS process is in question, or a comparator which triggers in the required voltage range and drives a switch. The

switches need to turn ON completely to offer minimal resistance to current in the result branch. For example, in the realization of an OR gate, the threshold range for the switch to activate/turn ON is between 0.07V to 0.13V. This is a very narrow range for any switch to work. It demands the use of a high gain comparator to activate the switch, adding to additional overhead. Also, due to noise, process variations (offsets induced in the threshold switch, change in the threshold range due to changes in resistance of memristors/switches, etc.) the idea may no longer be feasible. These issues have not been described in the literature.

Memristor models are used to simulate and obtain the delay and energy numbers. The model for the memristor is exactly the same as in [29]. A VTEAM model with Biolek window function is used with the following parameters. k_{on} = -216m/sec, k_{off} = 0.091m/sec, $V_{T,ON}$ = -1.5V, $V_{T,OFF}$ = 0.3V, x_{on} = 0, x_{off} = 3mm, a_{on} = 4, a_{off} = 4, R_{ON} = 1k ohm, R_{OFF} = 300k ohm. The transistors used are thick oxide devices from a 180nm planar CMOS process. Better performance can be expected with shorter channel length transistors.

A comparison of the delay/hardware required and energy is given below. As explained before, the numbers for MAD and MAGIC gates do not include the overhead associated in realizing the gates. If the overhead for threshold switches are included, the energy and hardware count number will significantly increase.

From Table 14, it is clear that the work presented has minimum possible latency of 1 cycle per operation. This delay does not include the set/reset/input initialization cycles like the other works. As explained before, the set/reset operations can all run in parallel at the time inputs are initialized. Table 15 compares the hardware required for the logic operations. It can be seen that the memristor count is very similar to that of MAD/MAGIC gates. The design presented here does not require any resistors as it uses transistors as current mirrors instead. The transistor count for MAGIC and MAD gates are not published.

It can be clearly concluded from their designs that the numbers will be significantly higher.

Table 16 compares the energy consumption.

Table 14: Gate delay comparison

Gate	Imply	MAD	MAGIC	Current Work
NOT	1	1	1	1
AND	3	1	1	1
NAND	2	1	1	1
OR	3	1	1	1
NOR	4	1	1	1
XOR	5	1	N/A	1
Copy	2	N/A	N/A	1

Table 15: Hardware comparison (Memristor, Resistor, Transistor)

Gate	Imply	MAD	MAGIC	Current Work	CMOS
NOT	2, 1, X	2, 2, X	2, X, X	2, 0, 2	0,0,2
AND	4, 1, X	3, 2, X	3, X, X	3, 0, 4	0,0,6
NAND	3, 1, X	3, 2, X	3, X, X	3, 0, 4	0,0,4
OR	5, 2, X	3, 2, X	3, X, X	3, 0, 4	0,0,6
NOR	5, 2, X	3, 3, X	3, X, X	3, 0, 4	0,0,4
XOR	6, 3, X	3, 2, X	N/A	3, 0, 12	0,0,16
Copy	2, 1, X	N/A	N/A	2, 0, 2	0,0,4

X – Information incomplete, numbers are not reported in published literature.

Table 16: Energy comparison (Joules)

Gate	Imply	MAD	Current work	CMOS
NOT	2.7e-14	3e-14	2.7e-14	1.06e-14
AND	8.1e-14	3e-14	2.7e-14	2.14e-14
NAND	5.4e-14	3e-14	2.7e-14	1.09e-14
OR	8.1e-14	3e-14	5.4e-14	2.16e-14
NOR	10.8e-14	3e-14	5.4e-14	1.07e-14
XOR	13.5e-14	3e-14	7.2e-14	4.4e-14
Copy	5.4e-14	N/A	2.7e-14	2.1e-14

Resistors are usually larger than MOS transistors. In this technique, since the MOS transistors are used as active devices (and not just as mere switches), there is a significant area advantage of at least 3x-4x. By using current mirrors, we can control the speed of operation (delay), power and area by controlling a single parameter which is the width of the CMOS transistor devices. This helps ease the complexity for synthesis algorithms. From a manufacturability standpoint, the following can be deduced.

- Imply gates are complex. It takes up large area. This presents significant challenges in having low routing resistances to connect different gates. In short channel technologies, the via resistance also becomes dominant in multi-layer ICs. This can present a challenge at the fabrication end to have thicker metal layers or metals with low resistivity.
- MAD gates have a fundamental problem in implementing threshold switches. These needs to be CMOS compatible devices. The process needs tighter control to make these switches actuate in smaller voltage ranges.
- The premise of this research is to complement CMOS processing and not replace it. There will always be CMOS circuits in the background to perform computations. Current mirror based logic is CMOS friendly. Memristors built either on metal layers or on the substrate can readily be connected to already present CMOS devices.

Comparison of memristor based logic gates with CMOS is very difficult because of the very nature of the application. Memristor technology is too nascent compared to the mature CMOS processes. There are many variables in computing the energy numbers for a CMOS gate. Factors like the type of transistors chosen (whether it is high threshold voltage or low threshold voltage devices), channel length of the transistors, drive strength of the gates,

width of each device, operating voltage etc. to name a few. Memristor based designs do not have many variables. There is a very limited set of memristor models available in the scientific community for circuit designers to make a fair comparison.

4.3 DESIGN OF A FULL ADDER

A conventional full adder described in [33] consists of 2 XOR gates to compute the SUM, two AND gates and an OR gate to compute the carry. A, B, C_{in} are the inputs. S is the sum and C_{out} is the carry output. Figure 36 shows the input memristors M_A, M_B and M_{Cin} connected to V_1, V_2 and V_4. As explained before, in the zeroth cycle, the inputs are initialized by applying V_{SET} or V_{RESET} to the voltage sources. In the same cycle, M_C, M_D, M_E, M_{SUM}, M_{Cout} are all reset by applying the appropriate V_3, V_5.

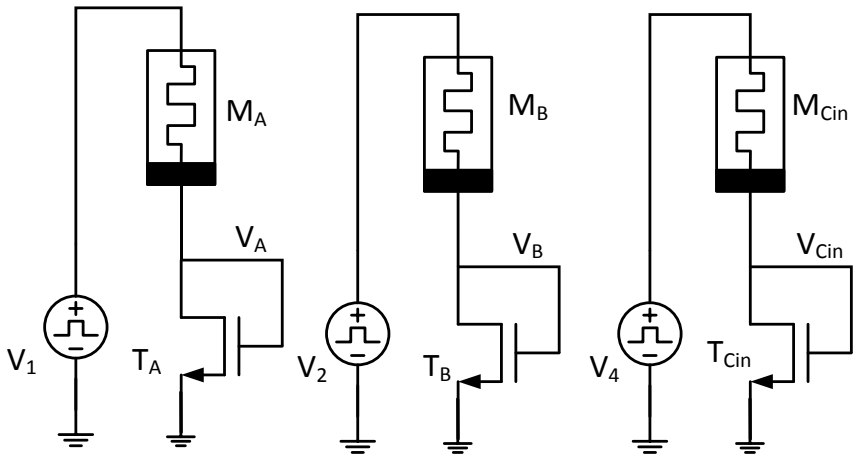


Figure 36: Input gates for the Full Adder

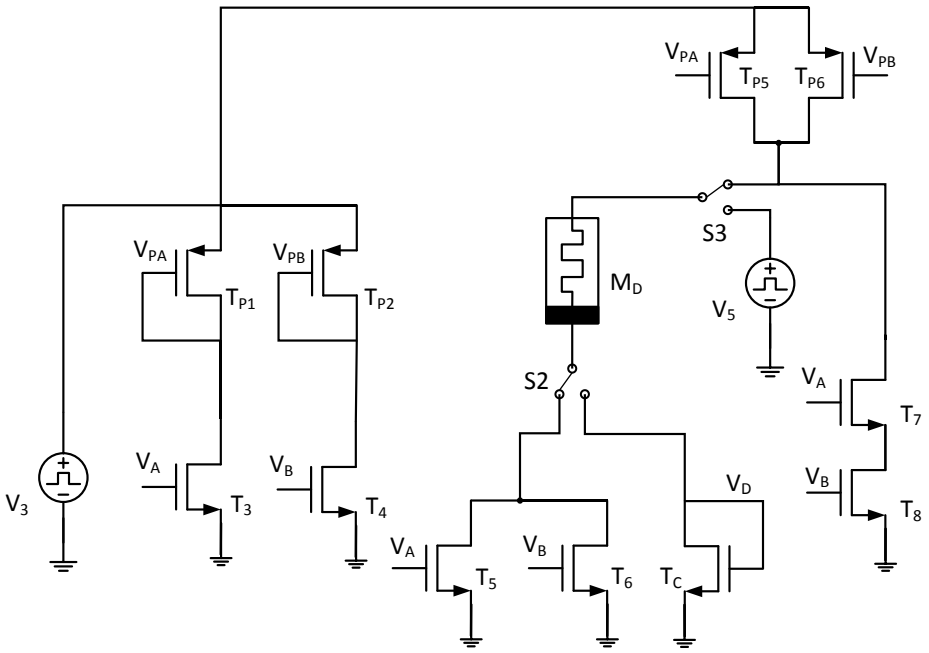


Figure 37: Intermediate SUM generation - XOR

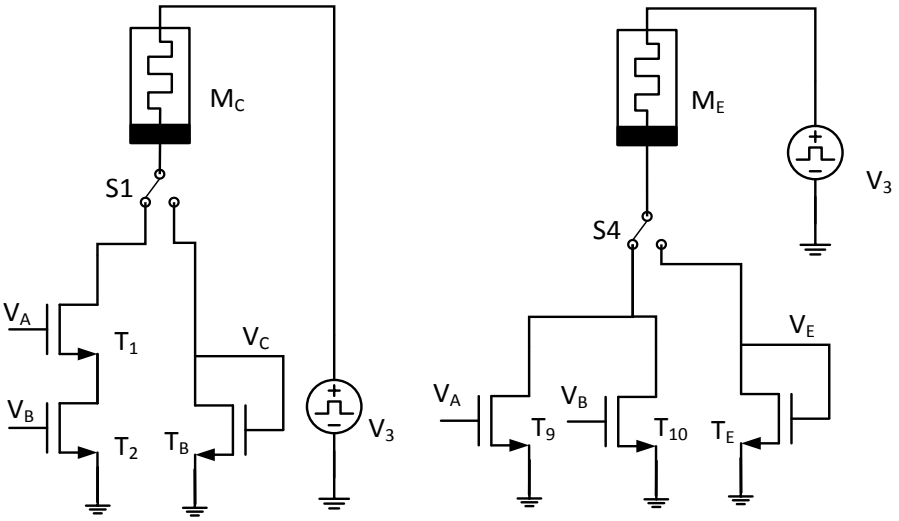


Figure 38: Intermediate carry generation – AND and OR gates

In the first cycle, the intermediate sum and intermediate carries are calculated using the circuits of Figures 7 and 8, respectively. In the first cycle, V_1, V_2 and V_4 are all set to V_{COND}. V_3 assumes a value of V_{SET}. The switch positions are as shown in the figures. An intermediate sum M_D which is the XOR of A and B is calculated. At the same time, an AND (M_C) and OR (M_E) operation between A and B is computed for the purpose of carry generation. It should be noted that the same V_3 is applied for all the 3 operations.

In the second cycle, these values are used to further generate the final sum and carry. Hence, switches depicted by S2 - S7 etc. are used.

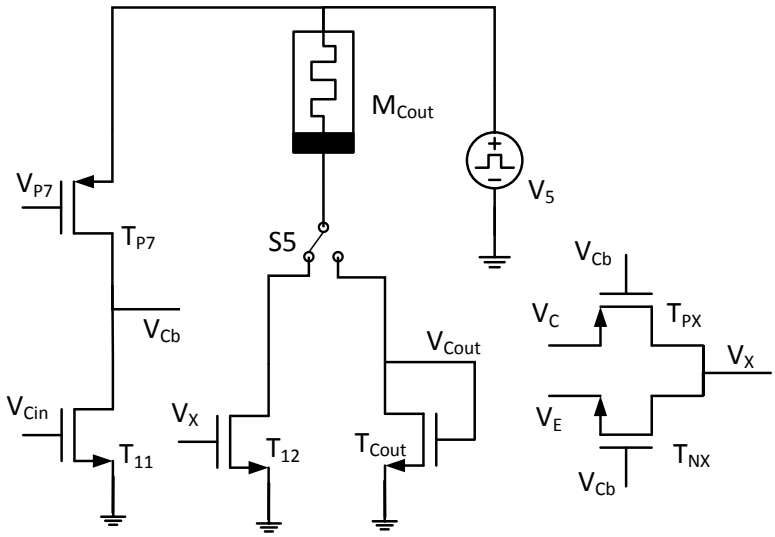


Figure 39: Final carry generation

In the second cycle, the final sum is computed using the XOR gate similar to the one shown in Figure 37. The carry is computed using the circuit shown in Figure 39. For this purpose, the states of the intermediate sum (M_D), and intermediate carry inputs (M_C, M_E) are required. The switches change the position. V_3 now assumes the value of V_{COND}. V_5 assumes a value of V_{SET}. The calculation of M_{Sum} is straightforward as it is an XOR

between M_D and $C_{in.}$ The carry generation can be analyzed with the equation $C_{out} = AB + C_{in}(A+B)$. It can be concluded that C_{in} is a gating element determining whether C_{out} is AB or A+B. Hence, in the first cycle, both AB and A+B were calculated. Depending on whether C_{in} is 0 or 1, the pass-gate switches shown in Figure 39 allows V_C (AND gate output) or V_E (OR gate output) to be connected to the final carry-out memristor.

The sum and carry-out are available in 2 cycles. The design consumes 8 memristors (including the input memristors) and 51 transistors. It needs only 2 different clocks (V_1 and V_3) since the rest of the clocks are phase shifted versions of the same reference clock.

4.4 RIPPLE CARRY ADDER

A ripple carry adder is a cascade of the full adders described in the previous section. The carry propagation is similar to that of a full adder. It can be analyzed with the equation $C_n = A_nB_n + C_{n-1}(A_n+B_n)$. Since C_{n-1} is the gating element determining whether C_n is A_nB_n or A_n+B_n, the operation becomes very similar to the carry calculation of a full adder. Thereby, the carry propagation delay per stage is reduced to just 1 cycle. For an N-bit ripple carry adder, the delay to the final sum is calculated as follows.

(A_1, B_1) to C_1 delay = 2 cycles
# of intermediate stages = N-2
C_1 to C_{N-1} delay = N – 2 cycles
C_{N-1} to S_N delay = 1 cycle

Total delay to sum = N+1 cycles
Total delay to carry = N+1 cycles

The area requirements for an N-bit ripple carry adder is as follows.
# of memristors = 8N
Transistors = 51N

Table 17: RCA complexity comparison (delay and hardware)

N bit	Imply	MAD	Hybrid-CMOS	Current Work
Delay	2N+8	N+1	2N+4	N+1
Memristors	11N	8N	14N	8N
Transistors	8N-1	14N	16N	51N
Resistors	N	9N	0	0
Drivers	0	2N+3	0	0

4.5 ARRAY MULTIPLIER

An array multiplier is shown in Figure 29. An N-bit array multiplier consists of N^2 AND gates, $N^2 - 2N$ full adders and N half adders. In an NxN array multiplier, it can be shown that that the critical path of the design consists of 1 AND gate delay, 2 half-adder delays (*input to sum* and *carry to carry* delay), 2N-2 full adder delays (N-2 *input to sum* delays and N-1 *carry to carry* delays and 1 *carry to sum* delay).

Every input bit (a_i, b_i) drives N different AND gates. With implication logic, the concurrency issues limit these operations and they cannot run in parallel. The innovation presented in this paper which uses current mirrors with memristors helps overcome this fundamental problem.

The half adder and full adders are designed as explained before. With this, the total delay to the MSB S_N can be shown as follows.

AND gate delay = 1 cycle
# of half adders = 2
Half adder delay = 2x1 cycle
# of full adders = 2N-2
Full adder delay = (N-2)x2 + (N-1)x1 + 1

Total input to MSB delay = **3N – 1**

The hardware requirement for an array multiplier is as follows. Each AND gate needs 1 memristor, full adders need 8 memristors, half adders need 4 memristors. Since the execution is sequential, it is possible to reuse the memristors used in previous cycles. For an NxN array multiplier, there are N-1 stages of full adders. Every stage has N-1 full adders and requires **8(N-1)** memristors. It can be shown that all the other stages can be implemented with 2x the memristors needed per stage. Once the computation is completed in an i^{th} stage, the memristors can be reset and used for $i+2^{th}$ stage. This reuse significantly saves the required hardware.

# of AND gates = N^2
Memristors for AND gates = N^2
# of half adders = N
Memristors for half adders = 4N
# of full adders = N^2-2N
Memristors for full adders = 2 x 8(N-1)

Total memristors = $\mathbf{N^2 + 20N - 16}$

Table 18: Metrics of an array multiplier

Bits	Imply Logic		Current Work	
N	Delay	Memristors	Delay	Memristors
5	54	213	14	109
8	90	414	23	208
16	186	1214	47	560
32	378	3966	95	1648
64	762	14078	191	5360

4.6 SUMMARY

A novel technique of using memristors for logic applications with the help of CMOS current mirrors has been demonstrated. It has been shown that compared to

previous literature, the current design has better coverage of the design aspects of memristor based circuits. It has been shown that the design has better performance metrics in terms of hardware requirement, energy etc. A full adder design has also been explained and shown that the sum and carry have a delay of only 2 cycles. A ripple carry adder with better performance metrics compared to existing literature has been presented. It is shown that an N bit ripple carry adder takes only N+1 cycles to compute the final output. An array multiplier has also been implemented and the numbers are compared against implication logic based designs. It is very important to understand that the MOS transistors act as current sources and not as switches. This allows them to control the switching power, speed etc. If they were to act as switches, it would result in the loss of a control knob for controlling power. The logic circuits built with MOS devices used as switches would burn more current compared to the ones built with current sources in the context of memristor based logic. It has been shown that current mirror based designs offer higher performance, lower power, higher speeds and also easier to sequence due to lesser number of stages.

4.7 DESIGN CONSIDERATIONS

When designing memristor based logic circuits with current mirrors, the following factors should be considered.

- Operating voltages of memristors vary significantly. Some devices operate at high voltages (+/- 3V) and some at +/- 1V. CMOS transistors selected for the design must be able to support such high voltages to prevent oxide breakdown and adverse aging effects.
- Speed of operation is dependent on the compliance current limiting resistor, in this case, it is the sizing of the current mirror. The resistance is inversely proportional to

the width of the CMOS device. By having a larger sized mirror pair, faster switching can be achieved. But eventually, this will be limited by the capacitances introduced by the mirror pair themselves.

- In cases where 2 mirror pairs are in series, like in AND gates, it may be better to use PMOS and NMOS in combination to avoid mismatch in threshold voltages. Also, as explained before, the sizing of the NMOS transistors increases slightly.
- Off state leakage currents must be considered when designing these logic gates. If the leakage is sufficient to change the state of a memristor, it will result in false switching.

Chapter 5: Summary and Future work

Different designs of memristor implementation of fixed point adder architectures like ripple carry adder, carry look-ahead adder, conditional sum adder, carry select adder and different parallel prefix adder architectures is explained. Implication logic based designs have shown better metrics in terms of delay and area compared to other published literature on implication logic. To overcome some of the key disadvantages of material implication, a new high fan-out technique of using current mirrors has been explained. These designs provide better delay/area/power numbers compared to state of the art. It reduces the complexity of controlling and sequencing different branches of a logic circuit.

Spice simulation results with standard memristor and CMOS models are used to verify the designs presented here. The numbers reported w.r.t delay, energy and area are better than those from previously reported works.

The speed limitation with memristors eventually depends on the operating speed of the CMOS control logic. Currently, memristors have been designed to switch at higher speeds compared to CMOS. CMOS mirror logic inherently has higher operating speeds because of its bandwidth. Also, unlike conventional CMOS logic, mirror logic need not switch voltages from rail to rail.

4.1 Limitations and Key issues

Current mirror based memristors can have a higher operating speed. Current mirror logic is faster than convention CMOS logic, merely because of the fact that current mirrors have a higher bandwidth. But logic operations with memristors are sequential operations. There must be a control logic associated with the circuits to control the timing sequence as to when V_{SET} and V_{COND} are applied to different branches of a logic circuit. Although some

memristors are faster than CMOS in terms of switching speeds, they still need to be controlled by a CMOS backplane in the timing control block. This will eventually be the bottleneck, preventing the increase in the speed of computation.

Memristor technology is still in a nascent state. Although there are some memristors which operate at CMOS compatible range of +/- 1V, some devices require high operating voltages (as high as 4V). Conventional CMOS transistors might not be able to work at such high voltages. It forces the need to use thick-oxide devices, which inherently slows down the speed of operation of the logic gate. Thick oxide devices have lower drive strengths and need to be sized large to support a reasonable current. This increases the capacitance at the gate, slowing down the speed of operation.

As channel length reduces, the leakage in the transistors become predominant due to short channel effects. Memristors requiring a low compliance current can falsely switch states in such a situation. This has to be carefully understood and simulated.

4.2 FUTURE WORK

The research presented here has relied on simulation results of different memristor and CMOS transistor models. It is very important to get a proof of concept on silicon and demonstrate the performance of the design. One way to do this is to fabricate the CMOS backend which contains a) "Timing control circuit" controlling the sequencing and b) different current-mirror based logic circuits which will eventually drive the memristors. Once this IC chip is fabricated in conventional CMOS processes like 65/90/180nm, memristors can be implanted on top of the IC chip and tested. Stand-alone board level testing can also be done, but due to larger capacitances at the board level, the speed of operation will be limited.

The issue of higher operating voltage still needs to be solved. There is a need for circuit level innovations to use regular core devices (low voltage CMOS) to operate at higher voltages compatible with certain memristors.

Off-state leakage needs to be controlled. A simple solution would be to stack devices. This way, the transistor on top of the stack will see a negative V_{GS}, helping to reduce the off-state leakage current. However, this comes at a penalty of larger devices sizes. Although the increase in size can cause a linear increase in leakage, having a -ve V_{GS} creates an exponential effect on leakage reduction since MOS transistors in subthreshold region behaves more like a BJT. This needs to be investigated.

The designs presented assumes a bipolar memristive operation. There are numerous memristors which are unipolar, most of them being CMOS compatible. This opens new opportunities to design new circuits to work with unipolar memristors.

Appendix – Verilog-A model

The following is the memristor model used in simulations presented in this dissertation. The model was developed by Israel Institute of Technology [32].

```
`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);
module Memristor(p, n);
  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 = 4;
// define the model:
// 0 - Linear Ion Drift;
// 1 - Simmons Tunnel Barrier;
// 2 - Team model;
// 3 - Nonlinear Ion Drift model
// 4 - Vteam model;

   parameter real window_type=2;
// define the window type:
// 0 - No window;
// 1 - Jogelkar window;
// 2 - Biolek window;
```

```
// 3 - Prodromakis window;
// 4 - Kvatinsky window (Team model only)
// 5 - Kvatinsky window2 (Vteam model only)

  parameter real dt=1e-13;
// 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=1;
// the initial state condition [0:1]

////////////// Linear Ion Drift model ///////////////

 //parameters definitions and default values
   parameter real Roff = 300000;
   parameter real Ron = 1000;
   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 threshhold_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
//for Simmons Tunnel Barrier model
   parameter real c_off = 3.5e-6;
   parameter real c_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 = 4;
   parameter real a_off = 4;

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

////////////////TEAM model////////////////////

   parameter real K_on=-216.2;
   parameter real K_off=0.091;
   parameter real Alpha_on=3;
   parameter real Alpha_off=3;
   parameter real v_on=-1.5;
   parameter real v_off=0.3;
   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))<threshhold_voltage/R) begin
          w=w_last;
       dwdt=0;
      end

// No window
   if ((window_type==0)|| (window_type==4)) begin
```

```
            w=dwdt*dt+w_last;

          end // No window

        // Jogelkar window
           if (window_type==1) begin

            if (sign(I(p,n))==1) begin
               sign_multply=0;
               if(w<p_window_noise) begin
               sign_multply=1;
               end
            end
            if (sign(I(p,n))==-1) begin
                  sign_multply=0;
                  if(w>(D-p_window_noise)) begin
                     sign_multply=-1;
                        end
            end

            w=dwdt*dt*(1-pow(pow(2*w/D 1,2),p_coeff))+w_last
+sign_multply*p_window_noise;

          end // Jogelkar window


        // Biolek window
          if (window_type==2) begin

            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(pow(w/D-stp_multply,2),p_coeff))+w_last;

          end // Biolek window


         // Prodromakis window
          if (window_type==3) begin
```

```
            if (sign(I(p,n))==1) begin
               sign_multply=0;
               if(w<p_window_noise) begin
               sign_multply=1;
               end
            end
            if (sign(I(p,n))==-1) begin
                  sign_multply=0;
                  if(w>(D-p_window_noise)) 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=c_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 =c_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

        // No window
          if (window_type==0) begin

          x=x_last+dt*dxdt;

          end // No window

          // Jogelkar window
        if (window_type==1) begin

                if (sign(I(p,n))==1) begin
                sign_multply=0;
                if(x<p_window_noise) begin
                        sign_multply=1;
                end
          end
          if (sign(I(p,n))==-1) begin
                sign_multply=0;
                if(x>(D-p_window_noise)) begin
                        sign_multply=-1;
                end
                end

             x=x_last+dt*dxdt*(1-pow(pow((2*x_last/D-
1),2),p_coeff))+sign_multply*p_window_noise;

          end // Jogelkar window


        // Biolek window
          if (window_type==2) begin

             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(pow((x_last/D-stp_multply),2),p_coeff));

        end  // Biolek window

    // Prodromakis window
        if (window_type==3) begin

            if (sign(I(p,n))==1) begin
            sign_multply=0;
            if(x<p_window_noise) begin
                sign_multply=1;
            end
        end
        if (sign(I(p,n))==-1) begin
            sign_multply=0;
            if(x>(D-p_window_noise)) begin
                sign_multply=-1;
            end
            end
            x=x_last+dt*dxdt*J*(1-pow((pow((x_last/D-
0.5),2)+0.75),p_coeff))+sign_multply*p_window_noise;

        end  // Prodromakis window


    //Kvatinsky window
        if (window_type==4) begin

            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);


// No window
    if ((window_type==0) || (window_type==4)) begin
       w=w_last+dt*dwdt;
```

```
        end // No window

      // Jogelkar window
      if (window_type==1) begin

            if (sign(I(p,n))==1) begin
            sign_multply=0;
            if(w<p_window_noise) begin
                  sign_multply=1;
            end
        end
        if (sign(I(p,n))==-1) begin
            sign_multply=0;
            if(w>(D-p_window_noise)) begin
                  sign_multply=-1;
            end
            end

          w=w_last+dt*dwdt*(1-pow(pow((2*w_last-
1),2),p_coeff))+sign_multply*p_window_noise;

        end // Jogelkar window


     // Biolek window
        if (window_type==2) begin

          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(pow((w_last-stp_multply),2),p_coeff));

        end // Biolek window


     // Prodromakis window
        if (window_type==3) begin

            if (sign(I(p,n))==1) begin
```

```
                sign_multply=0;
                if(w<p_window_noise) begin
                        sign_multply=1;
                end
            end
            if (sign(I(p,n))==-1) begin
                sign_multply=0;
                if(w>(D-p_window_noise)) begin
                        sign_multply=-1;
                end
                end

                w=w_last+dt*dwdt*J*(1-pow((pow((w_last-
0.5),2)+0.75),p_coeff))+sign_multply*p_window_noise;

            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))<threshhold_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
```

```
//////////////////// VTEAM model //////////////////

 if (model==4) begin // VTEAM model

   if (V(p,n) >= v_off) begin
      dxdt =K_off*pow((V(p,n)/v_off-1),Alpha_off);
   end

   if (V(p,n) <= v_on) begin
      dxdt =K_on*pow((V(p,n)/v_on-1),Alpha_on);
   end

      if ((v_on<V(p,n)) && (V(p,n)<v_off)) begin
   dxdt=0;
   end


// No window
   if (window_type==0) begin

   x=x_last+dt*dxdt;

   end // No window

   // Jogelkar window
if (window_type==1) begin

         if (sign(V(p,n))==1) begin
         sign_multply=0;
         if(x<p_window_noise) begin
                  sign_multply=1;
         end
   end
   if (sign(V(p,n))==-1) begin
         sign_multply=0;
         if(x>(D-p_window_noise)) begin
                  sign_multply=-1;
         end
         end


      x=x_last+dt*dxdt*(1-pow(pow((2*x_last/D-
1),2),p_coeff))+sign_multply*p_window_noise;
```

```
    end // Jogelkar window


  // Biolek window
    if (window_type==2) begin

      if (stp(-V(p,n))==1) begin
        stp_multply=1;
      end
      if (stp(-V(p,n))==0) begin
        stp_multply=0;
        end

      x=x_last+dt*dxdt*(1-pow(pow((x_last/D-stp_multply),2),p_coeff));

    end  // Biolek window


  // Prodromakis window
    if (window_type==3) begin


        if (sign(V(p,n))==1) begin
        sign_multply=0;
        if(x<p_window_noise) begin
                sign_multply=1;
        end
    end
    if (sign(V(p,n))==-1) begin
        sign_multply=0;
        if(x>(D-p_window_noise)) begin
                sign_multply=-1;
        end
        end

        x=x_last+dt*dxdt*J*(1-pow((pow((x_last/D-
0.5),2)+0.75),p_coeff))+sign_multply*p_window_noise;

    end  // Prodromakis window


  //Kvatinsky window2 VTEAM only
```

```
    if (window_type==5) begin

          if (V(p,n) >= 0) begin
          x=x_last+dt*dxdt*exp(-exp((x_last-a_off)/x_c));
          end

       if (V(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 VTEAM model
 end   // end analog

endmodule
```

References

[1] L. O. Chua, "Memristor-The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507-519, 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 1, 2008.

[3] 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 8, 2010.

[4] K. Bickerstaff and E. E. Swartzlander, Jr., "Memristor-Based Arithmetic," *Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*, 2010, pp. 1173-1177, Nov. 2010.

[5] Divya Mahajan, Matheen Musaddiq and E. E. Swartzlander, Jr. "Memristor Based Adders," *Conference Record of the Forty Eighth Asilomar Conference on Signals, Systems and Computers*, 2014, pp. 1256-1260, Nov. 2014.

[6] A. Adamatzky and L. Chua, *Memristor Networks*, Springer International, 2014.

[7] S. Kvatinsky, G. Satat, N. Wald, 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) Systems*, vol. 22, pp. 2054-2066, 2014.

[8] S. Shin, K. Kim and Kang, "Reconfigurable Stateful NOR Gate for Large-scale Logic-Array Integrations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 442-446.

[9] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the Association of Computing Machinery*, vol. 27, no. 4, pp. 831-838, 1980.

[10] M. Ziegler and M. Stan, "CMOS/Nano Co-Design for Crossbar-Based Molecular Electronics Systems," *IEEE Transactions on Nanotechnology*, December 2003

[11] S. Kvatinsky, M. Ramadan, E. G. Friedman, A. Kolodny, "VTEAM A General Model for Voltage Controlled Memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 62, pp. 786-790, Aug. 2015.

[12] Q. Zhu *et.al*, "A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing," *2013 IEEE International 3D Systems Integration Conference*, October 2013.

[13] D. Pala *et.al*, "Logic-In-Memory Architecture Made Real," *2015 IEEE International Symposium on Circuits and Systems*, 2015

[14] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the Association of Computing Machinery*, vol. 27, no. 4, pp. 831-838, 1980.

[15] T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders," *Proceedings 8th Symposium on Computer Arithmetic,* pp. 49-56, 1987.

[16] T. Breuer, A. Siemon, E. Linn, S. Menzel, R. Waser and V. Rana, "A HfO_2- Based Complementary Switching Crossbar Adder," *Advanced Electronic Materials*.

[17] J. H. Poikonen, E. Lehtonen and M. Laiho, "On Synthesis of Boolean Expressions for Memristive Devices Using Sequential Implication Logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31(7), pp. 1129-1134.

[18] F. Kim *et al.*, "Field Programmable Stateful Logic Array," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 30, no. 12, 2011.

[19] Q. Zhu et.al, "Design Automation Framework for Application Specific Logic-In-Memory Blocks," *Proceedings of 23rd IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2012.

[20] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nature* Materials, 2007.

[21] S. Balatti *et al*, "Normally-off Logic Based on Resistive Switches-Part II: Logic Circuits," *IEEE Transactions on Electron Devices*, vol. 62, no. 6, June 2015

[22] D. Stewart *et.al*, "Memristive Devices for Computing," *Nature Nanotechnology,* December 2012.

[23] Siemon *et al*. "In-Memory Adder Functionality in 1S1R Arrays," *IEEE ISCAS* 2015, 1338-1341.

[24] E. Linn, R. Rosezin, S. Tappertzhofen and R. Waser, "Beyond Von Neumann--Logic Operations in Passive Crossbar Arrays Alongside Memory Operations," *Nanotechnology*, 23(30), 305205.

[25] E. Lehtonen et al., "Recusive Algorithms in Memristive Logic Arrays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, 2015.

[26] S. Kvatinsky, G. Satat, N. Wald, 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) Systems*, vol. 22, pp. 2054-2066, 2014.

[27] L. Guckert and E. E. Swartzlander, Jr., "MAD Gates – Memristor Logic Design Using Driver Circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Dec 2015.

[28] Leon Chua, "Device Modeling Via Basic Nonlinear Circuit Elements," *IEEE Transactions on Circuits and Systems*, *CAS-27,* 1014–1044 (1980)

[29] S. Kvatinsky, M. Ramadan, E. G. Friedman, A. Kolodny, "VTEAM A General Model for Voltage Controlled Memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 62, pp. 786-790, Aug. 2015.

[30] 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.

[31] Jack C. Lee, *et al.,* "Understanding the Resistive Switching Characteristics and Mechanism in Active SiO_x Based Resistive Switching Memory," *Journal of Applied Physics*, October 2012.

[32] L. Xie *et.al*, "Fast Boolean Logic Mapped on Memristor Crossbar," *Proceedings of 33rd IEEE International Conference on Computer Design (ICCD)*, 2015

[33] N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd Edition, Reading, MA: Addison-Wesley Publishing Co., 1993.

[34] A. Weinberger and J. L. Smith, "A logic for High-Speed Addition," *National Bureau of Standards Circular*, No. 591, pp. 3-12, 1958.

[35] O. J. Bedrij, "Carry-Select Adder," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 340-346, 1962.

[36] M. Lehman and N. Burla, "Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 691-698, 1961.

[37] J. Sklansky, "Conditional-Sum Addition Logic," *IRE Transactions on Electronic Computers*, vol. EC-9, pp. 226-231, 1960.

[38] B. Razavi, *Design of Analog CMOS Integrated Circuits*, New York: McGraw-Hill, 2001.

[39] 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*, Vol. 61, pp. 1-5, 2014.

[40] P. M. Kogge and H. S., Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. 22, pp. 786-793, 1973.

[41] A. H. Shaltoot and A. H. Madian, "Memristor Based Carry Look-Ahead Adder Architectures," *Proceedings of IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, (pp. 298-301), 2012.

[42] J. Yao *et.al*, "Resistive switches and memories from Silicon Oxide," *ACS Publications*, August 2010.

[43] David Varghese, "Memristor Based High Linear Range Differential Pair," *Proceedings of the IEEE International Conference on Communication, Circuits and Systems*, 2009.

[44] S.B. Shouraki, “Memristor Based Circuits for Performing Basic Arithmetic Operations,” *Proceedings of Computer Science*, Science Direct, 2011.

[45] E. Lehtonen, “Arithmetic Operations Within Memristor Based Analog Memory,” *Proceedings of IEEE 12th International workshop on Cellular Nanoscale networks and their applications*, 2010.

[46] M. Ahmadi, “Optimized Implementation of Memristor Based Full Adder by Material Implication Logic,” *Proceedings of the 21st IEEE International Conference on Electronics, Circuits and Systems*, 2014.

[47] A.G. Radwan, “Memristor Based Balanced Ternary Adder,” *Proceedings of the IEEE 25th International Conference on Microelectronics*, 2013.

[48] A. Madian, “Memristor Based Carry Lookahead Adder Architectures,” *Proceedings of IEEE 55th International Midwest Symposium on Circuits and Systems*, 2012.

[49] L. Yan, “A Survey on Memristor Modeling and Security Applications,” *Proceedings of the IEEE 16th International Symposium on Quality Electronics Design*, 2015.

[50] H. Yang, “Memristor Based Approximated Computation,” *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, 2013.

Vita

Nagaraja Revanna is a Ph.D. student in the Department of Electrical and Computer Engineering of the University of Texas at Austin. His research interests are in the area of Analog/Mixed Signal IC Design. His current research is focused on designing novel logic circuits with memristors. He secured a "Bachelor of Engineering" degree in Electronics and Communication Engineering in the year 2012 from RV College of Engineering, Bangalore, India. He secured a "Master of Science in Engineering" degree in 2014 from the University of Texas at Austin in Integrated Circuits and Systems, with a thesis titled "Low Frequency CMOS Sinusoidal Oscillator for Impedance Spectroscopy". While working with Dr. Earl Swartzlander in the Application Specific Processor Group on his Ph.D, he has been an Analog Circuit Design Engineer at Intel Corporation, Austin designing XTAL oscillators, voltage regulators, bandgaps and thermal sensors since 2016.

Email: nagarajamails@gmail.com

This dissertation was typed by Nagaraja Revanna.