Logic Design within Memristive Memories Using Memristor Aided loGIC (MAGIC)

Nishil Talati, Saransh Gupta, Pravin Mane, and Shahar Kvatinsky, *Member, IEEE*

***Abstract*—Realizing logic operations within passive crossbar memory arrays is a promising approach to enable novel computer architectures, different from conventional von Neumann architecture. Attractive candidates to enable such architectures are memristors, nonvolatile memory elements commonly used within a crossbar, that can also perform logic operations. In such novel architectures, data is stored and processed within the same entity, which we term as memristive *Memory Processing Unit* (*MPU*). In this paper, Memristor Aided loGIC (MAGIC) family is discussed with various design considerations and novel techniques to execute logic within an MPU. We present a novel resistive memory- the *transpose memory*, which adds additional functionality to the memristive memory, and compare it with a conventional memristive memory. A case study of an adder is presented to demonstrate the design issues discussed in the paper. We compare the proposed design techniques with memristive IMPLY logic in terms of speed, area, and energy. Our evaluation shows that the proposed MAGIC design is 2.4X faster and consumes 66.3% less energy as compared to IMPLY-based computing for N-bit addition within memristive crossbar memory. Additionally, we compare the proposed design with IMPLY logic family on ISCAS-85 benchmarks, which shows significant improvements in speed (2X) and energy (10X), with similar area.**

***Index Terms*—IMPLY, MAGIC, Memristor, memristive Memory Processing Unit (MPU), transpose memory, von Neumann architecture.**

I. Introduction

RELENTLESS technology migration to the nanometer regime over the past few decades has led to high capacity memory and storage systems. This aggressive scaling, however, negatively affects the cost, performance, and reliability of flash and DRAM technologies, resulting in an increased interest in alternative memory technologies and architectures. Recently memristors, originally proposed by Chua in 1971, have shown promising solutions to these design challenges, and thus, have emerged as a prime interest among researchers. In [1], Chua proposed a fourth fundamental passive circuit element, apart from resistor, inductor, and capacitor. Chua and Kang extended the theory of memristors to memristive systems in 1976 [2]. Memristors and memristive devices are two-terminal electronic devices with variable resistance (also called memristance). This resistance depends on the amount and direction of the charge passed though the device and is bounded by minimum and maximum limits (R_{ON} and R_{OFF}, respectively). In this paper, we use the terms memristor and memristive device interchangeably, for simplicity.

Several possible applications involving memristors have evolved, such as nonvolatile memories [3], where resistance serves to store digital data, and the use of memristors as logic elements [4]–[8]. Additionally, memristors with high adaptive thresholds can be used to mimic the higher order behavior of synapses and thus can be utilized efficiently in neuromorphic systems [9]–[11].

The versatile nature of a memristor exploits the possibility of moving beyond conventional von Neumann architecture, as it can be used as both memory and logic element. In von Neumann architecture for massive parallel applications, data transfer requires a wide data bus, long latency, and consumes relatively high power [8], [12]. In novel architectures using memristors, memory and logic operations are performed within the same crossbar structure, resulting in almost no data transfer and significant reduction in latency and power. Thus, these architectures are potentially suitable for massive parallel applications, where a vast amount of data needs to be processed.

Three basic concepts to allow logic operations inside passive crossbar arrays are discussed in [13]. One of the concepts relies on programmable interconnects. Several such approaches expand this idea to realize Programmable Logic Arrays (PLAs) [14] and Field Programmable Logic Arrays (FPGAs) [15], for an example a CMOL FPGA [16], [17]. The second concept is about using the passive crossbar memories as Look Up Tables (LUTs) [18]. The third approach introduces realization of Boolean functions using stateful logic, such as IMPLY [5], [19].

An improved memristive stateful logic is MAGIC [20]. The quantifiable advantages of this logic over IMPLY are that, it requires a lower number of supply voltages, supports more basic Boolean functions, and it does not require additional hardware to the crossbar (such as resistors in the case of IMPLY). In MAGIC, a separate memristor is dedicated to output, whereas in the case of IMPLY, one of the input memristors acts as an output memristor. Thus, one of the inputs is always destroyed in IMPLY execution, but all the inputs are preserved in MAGIC.

This paper investigates the use of MAGIC for logic within

Nishil Talati* and Shahar Kvatinsky** are with the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion - Israel Institute of Technology, Haifa, Israel, e-mails: (nishil.talatiwok@gmail.com*, shahar@ee.technion.ac.il**).

Saransh Gupta* and Pravin Mane** are with the Department of Electrical, Electronics, & Instrumentation Engineering, Birla Institute of Technology & Science (BITS), Pilani, K.K. Birla Goa Campus, INDIA, e-mails: (*saransh.203@gmail.com, **pravinmane@goa.bits-pilani.ac.in).

This research is partially supported by the DST-FIST grant (FIST SI no. 133) to the Department of Electrical, Electronics & Instrumentation; BITS Pilani, K.K. Birla Goa Campus, by Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and by the Viterbi Fellowship in the Technion Computer Engineering Center.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

a memristive Memory Processing Unit (MPU) and makes the following contributions:

- We present a novel memristive memory crossbar called transpose memory, which adds functionality to the memristive crossbar. We also propose novel techniques to execute MAGIC operations within it.
- We propose techniques to parallelize the MAGIC execution over multiple rows and columns; and present a solution to isolate unselected rows and columns to avoid the possibility of distortion of data by applying isolation voltages, which is different than half-select in write disturb operation [21].
- We show how the non-idealities, in terms of parasitic resistances of nanowires, change the logic execution by re-evaluating all the constraints to execute logic and isolate unselected rows and columns within memristive MPUs.
- We demonstrate algorithms for complete logic execution within memristive MPUs with an example of one-bit full adder.
- We extend our approaches to N-bit addition, compare them with previously proposed pure-logic implementation within memristive MPUs, and show advantages of proposed approaches in terms of speed and energy with no significant area overhead. We also present a comparison of these designs on ISCAS-85 benchmark circuits to show the advantages of the proposed techniques.

The rest of the paper is organized as follows. Section II describes memristor modeling and resistive memory crossbars, including the introduction of transpose memory. Section III discusses methods to design two basic memory-compatible MAGIC operations (*i.e.,* NOR and NOT) with their design constraints. This section also talks about the effect of parasitic resistance of non-ideal wires and presents techniques to overcome it. Section IV proposes algorithms for logic within memristive MPUs of conventional and transpose memories with a case study of an adder. In Section V, latency, area, and energy of MAGIC are evaluated and compared to previously proposed pure memristive logic techniques for N-bit addition operation and for ISCAS-85 benchmark circuits. The paper is concluded in Section VI.

II. Memristive Memory

Major limitations of commercially available non-volatile memory- flash memory include low endurance [22] and a decrease in yield and reliability as device geometries get smaller [23]. These limitations motivate the development of emerging non-volatile memory technologies, such as Conductive Bridging RAM (CBRAM), Resistive Random Access Memory (ReRAM or RRAM), Phase Change Memory (PCM), and Spin-Transfer Torque Magnetoresistive RAM (STT-RAM), which can be considered as memristors [24].

Memristive technologies are non-volatile and compatible with CMOS fabrication process [25]. Memristive devices are expected to have low switching energies and fast switching speeds. The read and write times can be as fast as 120 ps [26], [27]. The switching energy is assumed as low as 1 pJ [27]. The endurance limit of memristors is measured approximately as 10^{10} allowed write operations per cell [28] (except STT-RAM, where 10^{15} is achieved). This limit is likely to increase to 10^{15} [29]. Memristive devices are fabricated between two metals, which act as the top and bottom electrodes of a dielectric material [30]. Hence, memristors can be fabricated in the metal layers as part of a standard CMOS Back End of Line (BEOL) process. Memristive memories generally utilize a crossbar structure, which enables an extremely dense memory of $4F^2$, where F is the feature size. Digital data is represented in terms of its resistance, where LRS (low resistance state, R_{ON}) is logical '1' and HRS (high resistance state, R_{OFF}) is logical '0.' This section talks about the memristor model used in this paper and introduces conventional and transpose memristive memories.

A. Memristor Modeling

There are several memristor models proposed in the literature [31]–[37]. Recently, ThrEshold Adaptive Memristor (TEAM) model [37], which relies on the current as a threshold parameter, has gained attention due to its simplicity, generality, and flexibility. However, some memristive technologies exhibit threshold voltage rather than threshold current. Furthermore, certain memory and logic operations, including MAGIC, demand voltage as the threshold parameter [38]. In this paper, we use the Voltage ThrEshold Adaptive Memristor (VTEAM) model [38], which has similar advantages as TEAM model and fulfills the requirements for proper operation of MAGIC gates. Additionally, VTEAM model is sufficiently accurate and exhibits less than 1.5% relative root mean square error when compared with the experimental results of resistance switching of memristive materials such as *Pt-Hf-Ti based memristor* [39], *Ferroelectric memristor* [40], and *metallic nanowire memristor* [41].

In VTEAM model, the derivative of the state variable (x) is

$$\frac{dx}{dt} = \begin{cases} k_{off}\left(\frac{v(t)}{v_{off}} - 1\right)^{\alpha_{off}} f_{off}(x), & 0 < v_{off} < v, \\ 0, & v_{on} < v < v_{off}, \\ k_{on}\left(\frac{v(t)}{v_{on}} - 1\right)^{\alpha_{on}} f_{on}(x), & v < v_{on} < 0. \end{cases} \quad (1)$$

Here, $k_{off}, k_{on}, \alpha_{off}$, and α_{on} are the model fitting parameters, v_{off} and v_{on} are the threshold voltages, and $f_{off}(x)$ and $f_{on}(x)$ are the window functions, which constrain the state variable to $x \in [x_{on}, x_{off}]$. The current-voltage relationship is

$$i(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{x_{off} - x_{on}}(x - x_{on})\right]^{-1} v(t). \quad (2)$$

B. Conventional Memristive Memory Crossbar

Fig. 1 shows the schematic of a $k \times m$ memristive crossbar. To write logical '1' and '0' to a memristor, V_{SET} and V_{RESET} are applied, respectively, across it. These programming voltages should be above the threshold voltages of the memristor. Half-select voltages (*i.e.,* $|V_{SET}/2| < |v_{on}|$ and $|V_{RESET}/2| < |v_{off}|$) are applied to isolate memristors of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

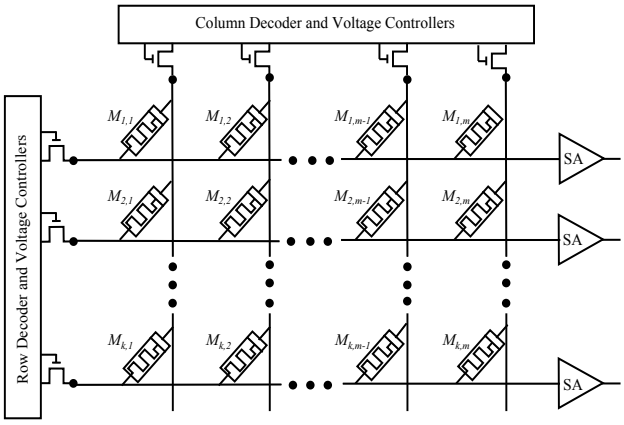

Fig. 1. Schematic of a $k \times m$ conventional memristive memory crossbar. Column (row) decoder is used to select a column (row) and voltage controllers assert various voltage levels on the selected column (row). SA represents the sense amplifier to sense the current in the orthogonal direction of applied voltage(s).

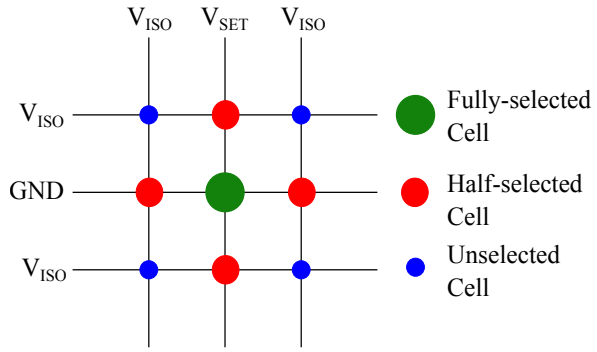

Fig. 2. Write disturb in memristive memory crossbar. Application of half the write voltage- $V_{SET}/2$ to unselected bit-lines and word-lines results in no applied voltage across unselected cells (marked in blue), while also preserving the resistance of the half-selected memristors (marked in red).

different row/column from which the data is being written due to the possibility of undesired write operations (which is also known as 'write disturb' [21]). Half-selected cells in a 3×3 memristive array are illustrated in Fig. 2. In the figure, a SET operation is being performed on the memristor located on the second row and the second column, marked in green. All the memristors in the second column are under the influence of V_{SET} (at one terminal) and all the memristors in the second row are under the influence of the ground voltage (at one terminal), and thus are half-selected. It is essential to bias the other terminals of the half-selected memristors at $V_{SET}/2$ to preserve the data (this is also known as $V/2$ biasing scheme [42]).

Read operation is achieved by applying V_{READ}, a voltage below the threshold level, across the selected memristor and measuring the current through the device using a sense amplifier (SA). One of the primary concerns associated with this operation is the sneak path phenomenon [43]–[46], which is an undesirable path for the current flow. This problem occurs because of the fact that read voltage produces additional current flow through paths, different than the desired one. This extra current flow adds resistance in parallel to the selected memristor, which depends upon the stored data in the unselected memristors. Several approaches are proposed to overcome this problem [43], [44], [47]. In this paper, we assume that these approaches are used to remove the sneak path problem. Note that the sneak path phenomenon restricts the array size. This bound depends on the non-linearity of the memristor model [48]. In this paper, we assume that the maximum size of the array is 512×512.

C. Transpose Memristor Memory Crossbar

Although the memristive memory crossbar structure is symmetrical, accessing memory cells in a conventional memory array is achieved only from one direction. The access from the other direction is blocked since only specific voltages can be applied in each row/column and the decoding and sensing circuits are connected to a single edge of the array. Additional peripheral circuitry would provide more flexibility to the memory array and would add capabilities to the memory system. We call this memory structure *transpose memory*. For example, transpose memory can be used to connect multiple processing units that access the array from different directions simultaneously [49]. In this paper, transpose memory is used to improve the logic functionality of memristive crossbars by enabling logical operations over columns as well, rather than solely over rows.

All operations (read, write, and half-selecting cells) are performed in transpose memory by application of similar voltages as in the conventional memory with the freedom of applying these voltages from both horizontal and vertical directions. During the read operation, only a single set of orthogonal directions (one for voltage application and the other for current sensing) is utilized at a time. Thus, the transpose memory architecture has the same number of sneak paths as conventional memory. However, only the direction of the sneak paths would be perpendicular if the other orthogonal set is used. Hence, similar techniques can be used to alleviate the sneak paths as conventional memory. In the transpose memory, each cell can therefore be sensed from both directions using two sets of peripheral circuitry to select and sense cells, as shown in Fig. 3. This memory is more suitable for large arrays, where most of the area is occupied by memory cells and the additional periphery circuitry can be located (at least partially) below the crossbar to save area [50]. The allowed set of voltages applied by the voltage controllers on each memory are listed in Table I. The last few rows in the table show the allowed voltages to perform MAGIC operation, as explained in Section III.

D. Overhead Associated with Transpose Memory

Additional functionality in the transpose memory comes at the cost of extra CMOS area. Fig. 4 illustrates the difference in peripheral circuitry between $k \times m$ conventional and transpose memory crossbars. The area remains same at the nanocrossbar layer(s), but it differs at the bottom CMOS layer(s). In this paper, a cross-coupled inverter latch-sense amplifier is used for current sensing, which requires seven CMOS transistors per nanowire; and voltage buffer is used for voltage application,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

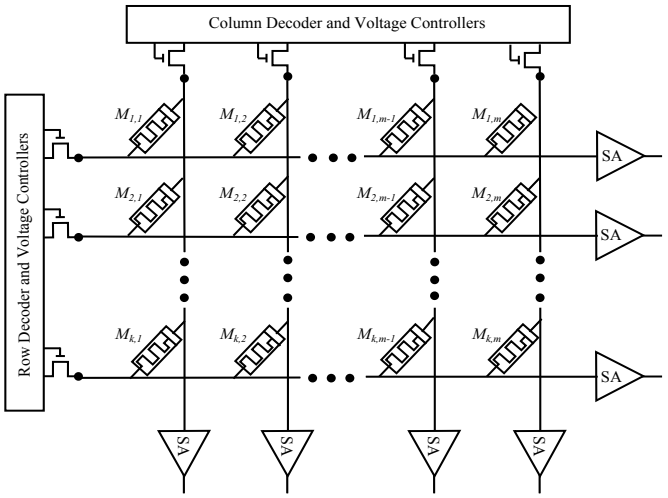

Fig. 3. Schematic of a $k \times m$ transpose memristive memory crossbar. Column (row) decoder is used to select a column (row) and voltage controller applies various allowed voltages on the selected column (row). Unlike conventional memory, in transpose memory the same voltages can be applied to both rows and columns. SA represents the sense amplifier to sense the current.

TABLE I
ALLOWED MEMORY OPERATIONS AND THE ASSOCIATED APPLIED VOLTAGES

Type	Operation	Applied Voltages
Memory	Write	V_{SET}, V_{RESET}
	Read	V_{READ}
	Ground	GND
	Half-Select	$V_{SET}/2$, $V_{RESET}/2$
MAGIC	Execute	V_0
	Row Isolate	V_{HS}
	Column Isolate	V_{VS} (only for transpose memory)

which costs four CMOS transistors per nanowire. Note that this comparison includes only the overhead due to voltage controllers and sense amplifiers, assuming that additional circuitry has a similar trend.

The number of transistors utilized in the CMOS peripheral circuit in $k \times m$ conventional memory is $4k+7m$, where as in the case of transpose memory, it is $11(k+m)$. Furthermore, the number of memristors utilized in an array in both cases is $k \cdot m$. Hence, the CMOS overhead is strongly dependent on the array size (*i.e.,* k and m). For simplicity, we assume that $k = m$; thus the CMOS area occupied underneath the transpose memory crossbar would be twice the area as in the case of conventional memory crossbar (*i.e.,* $22k$ versus $11k$). Even with the double area overhead, it becomes insignificant as compared to memristive area for large array sizes (*i.e.,* $22k$ versus k^2). Fig. 5 shows the comparison of the ratio of total area utilized at CMOS and memristive layer for different values of array sizes (*i.e.,* $k \times k$). The comparison shows that the ratio is almost equal (which implies the area utilization) for large array sizes (*i.e.,* $k \geq 100$). Note that this is a general comparison irrespective of the memristor technology used, *i.e.,* without considering the maximum allowed array size (which is 512×512 in this paper).

While executing logic functions in transpose memory for a given clock cycle, one set of orthogonal sides is used, out of two, which is the same as in conventional memory. Thus, additional CMOS drivers and sense amplifiers do not add any overhead in terms of latency in transpose memory.

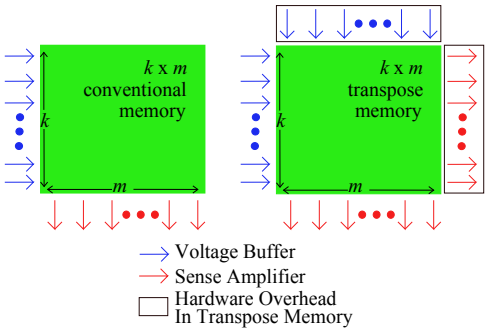

Fig. 4. Comparison of additional supporting CMOS circuitry to facilitate logic implementation at nanocrossbar layer for $k \times m$ conventional and transpose memories.

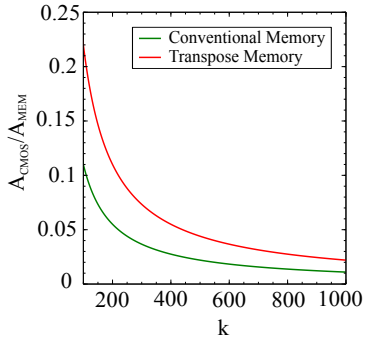

Fig. 5. Ratio between CMOS area (A_{CMOS}) and memristor area (A_{MEM}) for different array sizes (*i..e.,* different k for $k \times k$ arrays) for conventional and transpose memory crossbars. The area utilization at nanocrossbar layer increases for larger arrays.

III. MAGIC WITHIN MEMRISTIVE MPU

MAGIC is a stateful logic family, compatible for computation within memory [20]. In this logic family, to realize n-input Boolean functions (*i.e.* NOR, NAND, OR, AND, NOT), n input memristors and one output memristor are required. Among the MAGIC gates, NOR and NOT can be performed within a memristive memory crossbar due to the connection pattern among input and output memristors. In MAGIC NOR and NOT designs, the input(s) is (are) the data within the memristor memory and the output is the stored data after the computation. A regular read operation from the output memristor is carried out to sense the result of computation. Note that current memristive technologies suffer from a limited endurance of approximately 10^{12} writes per cell [28]. Executing logical functions within memory would increase the number of effective write operations to further stress memory cells, and thus, would decrease the lifetime of this memory. The limited endurance needs to be considered while executing logic operations within the memory. There are, however, projections that the endurance will be improved to higher values that would allow unlimited logical operations within memory [29].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

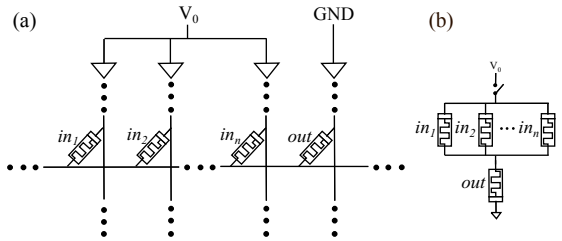

Fig. 6. (a) n-input NOR within a single crossbar row and (b) its circuit equivalent. Input memristors are in_1, in_2, ... in_n and an output memristor is out. MAGIC execution and ground voltages are represented as V_0 and GND respectively.

TABLE II
STEPS TO EXECUTE MAGIC NOR IN A ROW

Step #	Operation	Application of Voltages
1	Write R_{ON} at out	$out \longleftarrow V_{SET}$
2	Execute NOR operation	in_1, in_2, ... $in_n \longleftarrow V_0$ $out \longleftarrow GND$

As listed in Table I, three operations need to be added to support MAGIC execution within a memristive memory: MAGIC execute, row isolate, and column isolate (only for transpose memory). Incorporating them would increase the complexity of the CMOS peripheral circuit. The number of voltage levels increases from six to eight in the case of conventional memory; and to nine in the case of the transpose memory to support logic execution in addition to data storage. Thus, the design of analog multiplexers used to assert voltage remains the same in the case of conventional memory and expands by one additional selection bit in the case of transpose memory. Since the execution is carried out within the memory, we assume that the data would be present at appropriate memory locations before the execution, which removes the requirement of prior programming of inputs for all the versions of MAGIC as explained in this section.

This section describes the design and its corresponding constraints for MAGIC execution within a row and a column (for transpose memory) for the correct operation that preserves the input(s). Additionally, this section discusses the undesirable effect of MAGIC NOR on unselected cells and proposes a solution to eliminate this phenomenon. The effect of non-ideal wires is also incorporated into the discussion to examine the deviation of various circuit parameters from their ideal values.

A. MAGIC NOR within a row and a column

The schematic of an n-input NOR logic gate within a memristive MPU and its circuit equivalent are shown in Fig. 6. In this figure, in_1, in_2, ... in_n are the input memristors and out is the output memristor. There are two steps involved in the execution, as listed in Table II. Since the inputs are the stored data within the memory, they are not required to be written prior to the execution as a separate step.

Transpose memory opens up an opportunity to execute MAGIC NOR over both rows and columns. The execution over a column is slightly different than the row execution, as the input memristor(s) is (are) connected to ground and

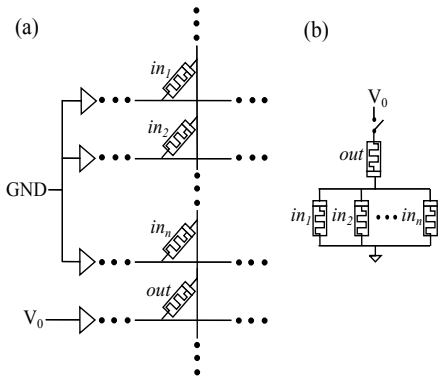

Fig. 7. (a) n-input NOR within a single transpose crossbar column and (b) its circuit equivalent. Input memristors are in_1, in_2, ... in_n and an output memristor is out. MAGIC execution and ground voltages are represented as V_0 and GND respectively.

TABLE III
STEPS TO EXECUTE MAGIC NOR WITHIN A COLUMN

Step #	Operation	Application of Voltages
1	Write R_{ON} at out	$out \longleftarrow V_{SET}$
2	Execute NOR operation	in_1, in_2, ... $in_n \longleftarrow GND$ $out \longleftarrow V_0$

the execution voltage V_0 is applied to the output memristor, as shown in Fig. 7. The steps involved in the execution of this logic are listed in Table III. Unfortunately, executing multiple NOR operations within the same row (or column) simultaneously is impossible, as illustrated in Fig. 8. Due to the connection pattern, two different NOR operations are not distinguishable and the output memristors of both operations are actually connected in parallel, leaving the equivalent resistance at the output $R_{ON}/2$, rather than R_{ON}, resulting in the wrong operation. However, to improve performance, it is possible to parallelize the operation over multiple rows (columns) as further explained in Section III-C.

B. Analysis and Evaluation of MAGIC NOR

The choice of execution voltage, V_0, is an important decision for correct and non-destructive MAGIC NOR operation. For correct circuit operation, the voltage across the output memristor, V_{out}, should be lower than v_{off}, when all inputs are logical zero, and greater than v_{off} for all other input combinations. The minimum value of V_{out} is determined by the case where one input is logical one and the rest of the inputs are logical zero. Additionally, when one of the inputs is logical zero, it is possible that the input would switch unintentionally to logical one if V_0 is above a certain value. To eliminate this undesired effect and have a non-destructive operation, the voltage across the input memristors is required to be lower than v_{on}. Thus, the allowed range of V_0 for proper execution of an *n*-input NOR is

$$\frac{v_{off}}{R_{ON}} \cdot \left\{ R_{ON} + \left(\frac{R_{OFF}}{n-1} \right) || R_{ON} \right\} < V_0, \tag{3a}$$

$$V_0 < min\left[v_{off} \cdot \left(1 + \frac{R_{OFF}}{nR_{ON}} \right), |v_{on}| \cdot \left(1 + \frac{nR_{ON}}{R_{OFF}} \right) \right]. \tag{3b}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

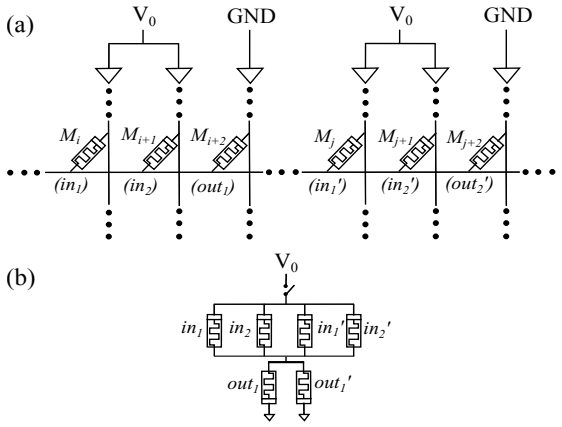

Fig. 8. Execution of multiple (two in this case) NOR operations in a single row: (a) at crossbar level, (b) equivalent circuit level schematic. It is impossible to distinguish between two MAGIC NOR gates within a single row and as a result, an inappropriate execution may occur.

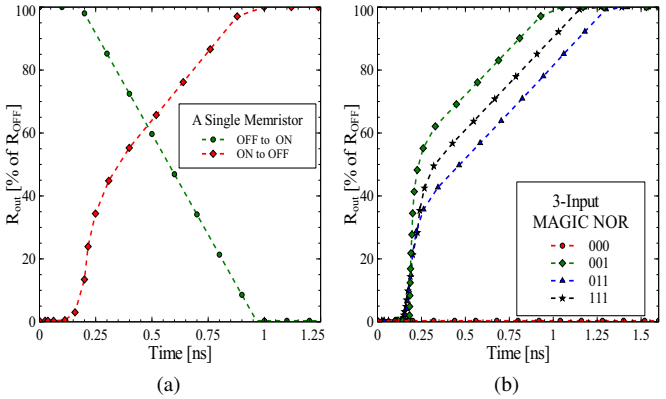

Fig. 9. SPICE simulations for (a) memristor switching time ($1ns$) for $V_{SET} = 2V$ and $V_{RESET} = 1V$, and (b) three-input MAGIC NOR delay for $V_0 = 1V$. The worst case delay is produced when one of the inputs is logical zero, which is $1.3ns$.

TABLE IV
MEMRISTOR PARAMETERS (FOR VTEAM MODEL [38])

Parameter	Value	Parameter	Value
R_{ON}	1 kΩ	x_{off}	3 nm
R_{OFF}	300 kΩ	k_{on}	-216.2 m/sec
v_{on}	-1.5 V	k_{off}	0.091 m/sec
v_{off}	0.3 V	α_{on}	4
x_{on}	0	α_{off}	4

Since MAGIC NOT is a special case of MAGIC NOR with $n = 1$, the constraint on V_0 for a NOT logic gate can be derived by substituting $n = 1$ in (3).

MAGIC NOR operations for multiple inputs are evaluated using the VTEAM model [38] for a $65nm$ CMOS process in Cadence Virtuoso. The model parameters of the memristor, as explained in Section II-A, are chosen to produce switching delay of $1ns$ for a voltage pulse of $1V$ of RESET and $2V$ of SET, and also to fit practical devices, as reported in [27]. The switching behavior of the memristor is shown in Fig. 9a. The memristor parameters are listed in Table IV.

The proposed architectures described in the following section (Sec. IV) incorporate MAGIC NOR with one to three inputs. Thus, the delay is evaluated for all cases and the worst case delay is considered as the deciding factor for the clock period of a memory cycle. From (3), the allowed values of V_0 are in the interval $[0.6V, 1.5V]$. The worst case delay is produced with three input MAGIC NOR having the logical values of inputs as {0,1,1}, {1,0,1}, and {1,1,0}, as shown in Fig. 9b. This delay is $1.3ns$ for $V_0 = 1V$, which is 30% more than the switching time of a single memristor. It is observed that increasing the value of V_0 decreases the delay of MAGIC NOR gate [20].

To determine the influence of memristor and CMOS process variations, we evaluate the alteration in the delay of three-input MAGIC NOR gates for various model parameters: α_{off}, k_{off}, R_{on}, and v_{off} and MAGIC execution voltage V_0. Our simulations show that α_{on} and k_{on} do not influence the delay. Furthermore, the delay remains unchanged if both R_{ON} and R_{OFF} are changed simultaneously keeping the R_{ON}/R_{OFF} ratio constant. Fig. 10 shows the variation of three-input NOR gate delay with respect to α_{off}, k_{off}, V_0, v_{off}, and R_{ON}. The simulations show that the delay decreases with an increase in α_{off}, k_{off}, and V_0, whereas the delay increases with an increase in R_{ON} and v_{off}.

C. Isolation of Unselected Cells During Parallel MAGIC Execution

When a two-input MAGIC NOR is executed over a row (Section III-A), V_0 is applied at V_i and V_{i+1} and GND is applied at V_{i+2} (Fig. 1). For all rows, the memristors lying on column i and $i + 1$ produce NOR outputs at the corresponding memristors on column $i + 2$. This technique enables computing multiple logical operations simultaneously in different rows, increasing parallelism of the execution. This is especially beneficial for applications with high data-level parallelism (DLP). The parallel execution becomes, however, an undesirable operation if it is necessary to preserve the data stored in unselected row(s), lying on the column V_{i+2}. Similarly, when the computation is carried out in a single column (Sec. III-A), all other columns are also affected. Thus, it is essential to isolate unselected rows/columns to stop undesirable NOR operation(s). This phenomenon is similar to write disturb in regular memristive memory operations [21].

Researchers are investing a wide range of efforts to solve the problem of isolating the unselected rows/columns while performing parallel execution. One of the attractive solutions is to instantiate selectors (for example, CMOS transistors) within memristive arrays, which completely removes the disturbance of unselected cells. However, this approach suffers from a significant decrease in the overall density of the memory structure. We propose to solve this problem by asserting isolation voltages on unselected rows/columns, which is similar to a half-select operation. Since the logic execution is also carried out by applying voltages, we believe that performing isolation using the same operation is an attractive solution for this problem.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

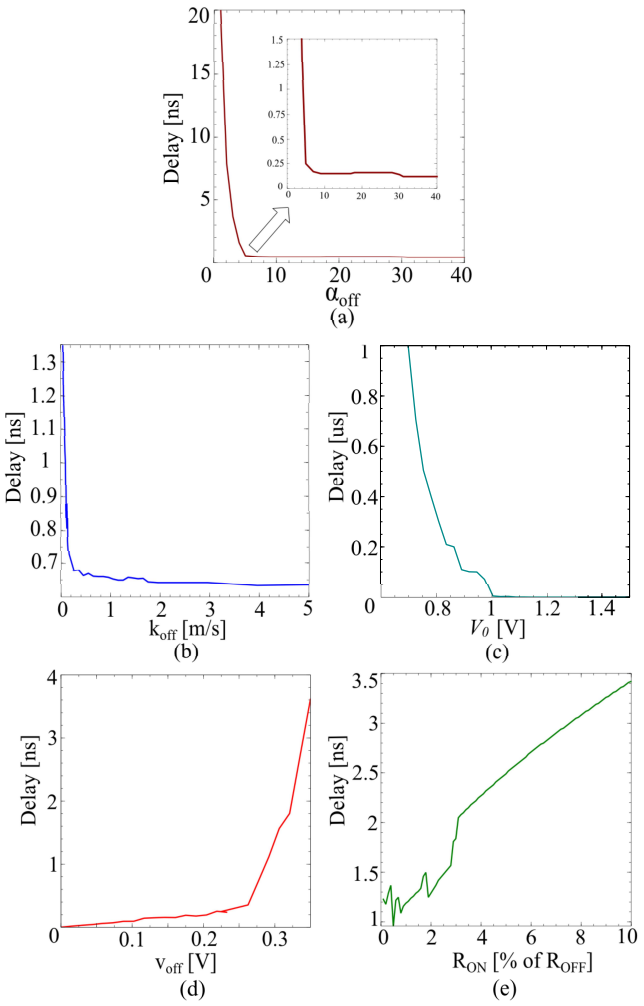

Fig. 10. Delay of a three-input MAGIC NOR gate for different values of (a) α_{off}, (b) k_{off}, (c) V_0, (d) v_{off}, and (e) R_{ON}. This delay is not influenced by α_{on} and k_{on}. The rest of the model parameters are as given in Table IV.

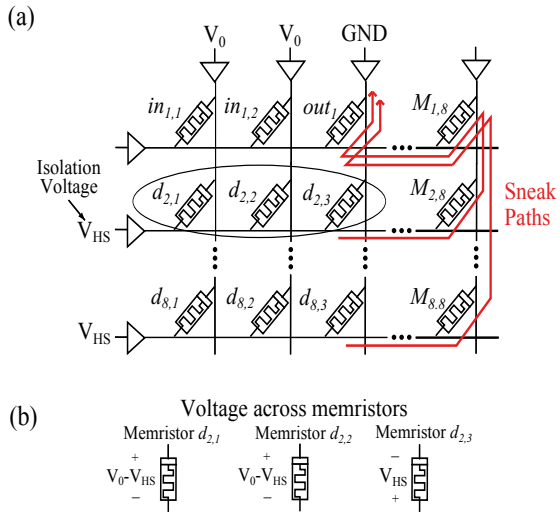

Fig. 11. An 8×8 array to demonstrate the isolation of unselected second to eighth rows while executing MAGIC NOR in the first row. (a) MAGIC NOR is the desired operation among the data present in the first row and the undesired for other rows. Isolation voltage V_{HS} is applied to prevent execution of MAGIC NOR in unselected rows. Isolation voltages produce sneak path currents as marked by red lines. (b) Resultant voltages across each memristor in the second row.

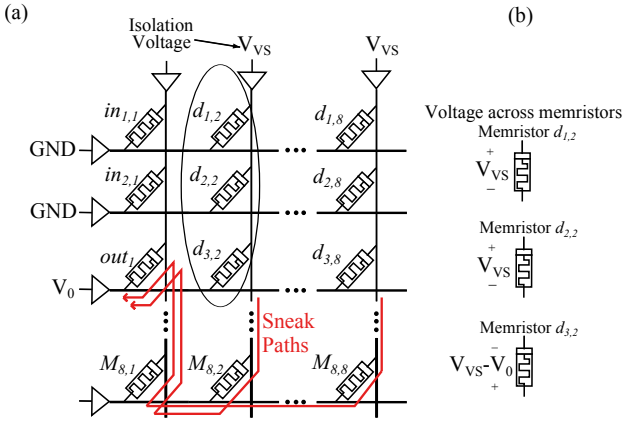

Fig. 12. An 8×8 array to demonstrate the isolation of unselected second to eighth columns while executing MAGIC NOR in the first column. (a) MAGIC NOR is the desired operation among the data present in the first column and the undesired for other columns. Isolation voltage V_{VS} is applied to prevent the execution of MAGIC NOR in unselected columns. Isolation voltages produce sneak path currents as marked by red lines. (b) Resultant voltages across each memristor in the second column.

While in the regular memory write disturb, the applied voltages are half of the write voltage values (*i.e.*, $V_{SET}/2$ or $V_{RESET}/2$), applying $V_0/2$ in MAGIC NOR would disturb the input memristors. Thus, as illustrated in Fig. 11, isolation of the unselected rows is carried out by the application of V_{HS} over unselected rows, which is

$$0 < |V_{HS}| < |v_{off}| < \frac{V_0}{2}. \tag{4}$$

Similarly, as illustrated in Fig. 12 for MAGIC over columns, V_{VS} is applied over unselected columns, which is

$$V_0 - |v_{off}| < |V_{VS}| < |v_{on}|. \tag{5}$$

The applied isolation voltage also produces current flow from the isolation voltage to ground through unselected memristors, as illustrated in Figs. 11 and 12. This *sneak path* current [43]–[46] does not change the state of any memristor since the isolation voltage is lower than threshold voltage. The output memristor is part of all sneak paths and therefore sneak path currents increase the current consumption of the gate. For current-controlled memristors [2], [37], the cumulative current may be higher than the threshold current, resulting in increased resistance of the output memristor, even when the logical state of the memristor is unchanged. This phenomenon is called the *state drift* [6] and does not exist in the voltage-controlled memristors considered in this paper.

To verify the isolation of unselected cells, an 8×8 crossbar structure is designed and evaluated in SPICE simulator with the circuit parameters listed in Table IV. In this experiment, execution of MAGIC NOR in a single row and isolation of the data present in all other rows is performed. In our evaluation, the logical values of all unselected cells is logical '1.' This configuration is used to determine the isolation voltage in the case of the worst potential unintended switching in unselected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

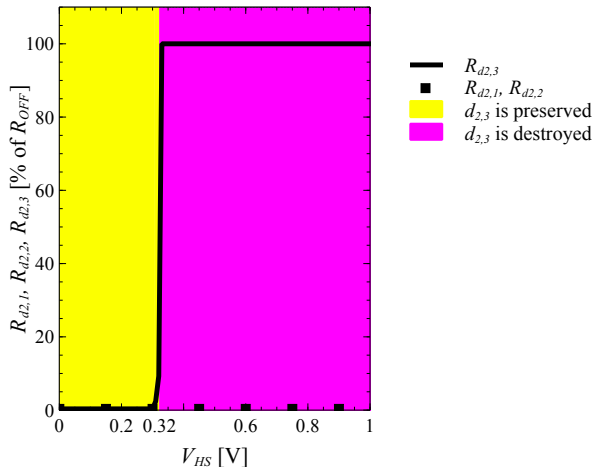

Fig. 13. SPICE simulation of resistance of $d_{2,1}$, $d_{2,2}$, and $d_{2,3}$ (Fig. 11) for different values of isolation voltage V_{HS} with $d_{2,1} = d_{2,2} = d_{2,3} = 1$. The logical state of $d_{2,3}$ is preserved for $V_{HS} < 0.32V$ and destroyed (pink region) for $V_{HS} > 0.32V$. The logical states of $d_{2,1}$ and $d_{2,2}$ are always preserved for any value of V_{HS}.

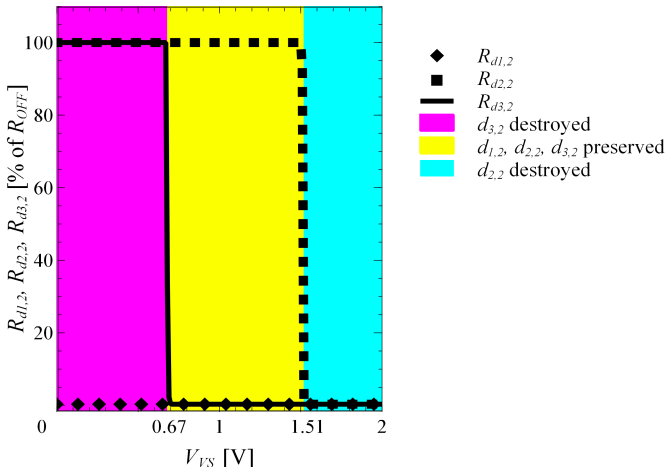

Fig. 14. SPICE simulation of resistance of $d_{1,2}$, $d_{2,2}$, and $d_{3,2}$ (Fig. 11) for different values of isolation voltage V_{VS} with $d_{1,2} = d_{3,2} = 1$ and $d_{2,2} = 0$. The logical state of $d_{3,2}$ is preserved for $V_{VS} > 0.67V$ and destroyed (pink shaded region) for $V_{VS} < 0.67V$. Similarly, the logical state of $d_{2,2}$ is saved for $V_{VS} < 1.51V$ and destroyed (cyan shaded region) for $V_{VS} > 1.51V$. The operation is always non destructive for $d_{1,2}$.

rows within column i+2. Additionally, in this configuration, all sneak paths flow to the output memristor producing the maximum sneak path current. While from (4), V_{HS} is in the range of $[0V, 0.3V]$, our results show that proper isolation is achieved for voltages of up to 0.32V, when allowing the state of $d_{2,3}$ to drift by 10%, as illustrated in Fig. 13.

Similarly, the proper isolation voltage for a MAGIC NOR execution over columns is evaluated with similar conditions. The primary difference between executions over columns as compared to rows is that the isolation voltage eliminates the possible destruction of all memristors (not only in row i+2). The upper bound of V_{VS} is determined by the state drift of $d_{2,2}$, when its value is initialized to logical '0' (all other memristors are set to logical '1'). The lower bound is determined as in a row operation. While from (5), V_{VS} is in a range of $[0.7V, 1.5V]$, our experimental results show that proper isolation is achieved for a wider range of $[0.67V, 1.51V]$, when allowing a state drift of 10%, as illustrated in Fig. 14.

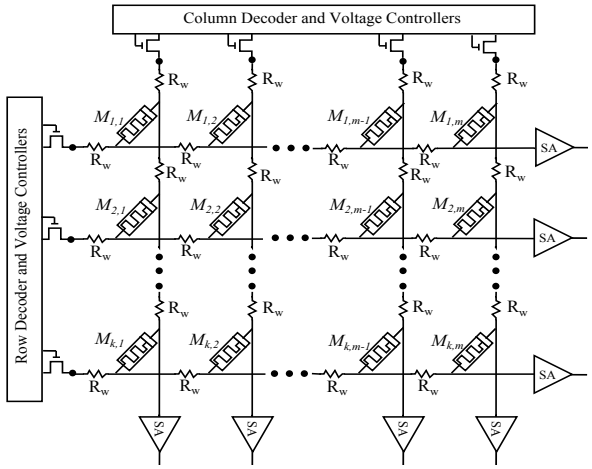

Fig. 15. $k \times m$ non-ideal transpose resistive memory crossbar. The resistance between each constitutive nanowires is R_w. Column (row) decoder is used to select a column (row) and voltage controller applies various allowed voltages on the selected column (row). SA represents the sense amplifier to sense the current.

D. Effect of Non Ideal Wires

In practice, crossbar nanowires possess parasitic resistance, that influence the required circuit parameters in memristive crossbar arrays. Fig. 15 shows a $k \times m$ transpose memory crossbar with non-ideal nanowires. The wire resistance depends on the length of the nanowire. Usually, the length between neighboring cells is identical for both rows and columns, and therefore, we assume that unit row and column wire resistances are equal to R_w. Note that the problems presented and solved in this subsection are similar to the ones presented in Sections III-B and III-C. While in the previous sections, the influence of the parasitic resistances of the nanowires is neglected, in this subsection, to deliver a realistic solution, we revisit these problems and propose the revised design considerations.

To determine the effect of wire resistance on V_0, assume all n-input memristors and an output memristor are situated within the same row, one next to the other, spanning the memory locations (i, j) to $(i, j+n)$. The equivalent circuit is shown in Fig. 16. We assume that $iR_w + R_m \gg R_w$ (where, R_m is the resistance of the memristor and can be either R_{ON} or R_{OFF}). Considering the effect of parasitic resistance of nanowires with the constraints described in Section III-B, the allowed range of V_0 for correct execution of n-input NOR is

$$\frac{v_{off}}{R_{ON}} \cdot \left\{ R'_{ON} + \left(\frac{R'_{OFF}}{n-1} \right) || R'_{ON} \right\} < V_0, \tag{6a}$$

$$V_0 < min\left[v_{off} \cdot \frac{\left(\frac{R'_{OFF}}{n} + R'_{ON} \right)}{R_{ON}}, |v_{on}| \cdot \frac{(R'_{OFF} + nR'_{ON})}{R_{OFF}} \right]. \tag{6b}$$

where R'_{ON} and R'_{OFF} denote the effective resistances and are equal to $(R_{ON} + iR_w)$ and $(R_{OFF} + iR_w)$ respectively. Note that these expressions are similar to (3). Fig. 17 shows

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

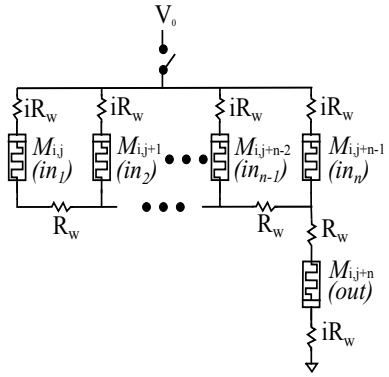

Fig. 16. Equivalent circuit of *n*-input MAGIC NOR over the row i of crossbar. in_1, in_2, ... in_n are input memristors and out is the output memristor.

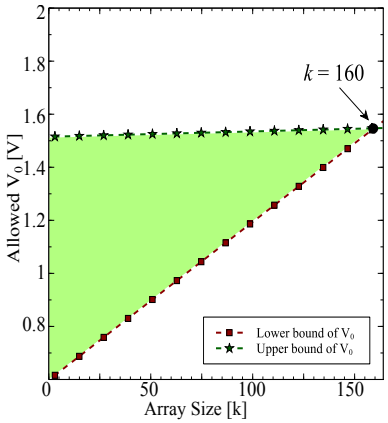

Fig. 17. Allowed V_0 for different array sizes ($k \times k$) with non-ideal wires ($R_w = 10\Omega$). It can be observed that the range of V_0 decreases with an increase in the array size. The largest array size possible for correct, non-destructive operation is 160×160.

the range of allowed V_0 for different array sizes ($k \times k$) for wire resistance R_w of 10Ω [51], [52]. The figure reveals that the allowed range of V_0 (shaded region) decreases for larger arrays, where the maximum array size for the proper operation of MAGIC NOR is 160×160.

Non-ideal nanowires also affect the value of the isolation voltages V_{HS} and V_{VS}, as illustrated in Fig. 18. To isolate the row i (column j), the allowed range of values of V_{HS} (V_{VS}) for preserving the logical state of d_1, d_2 and d_3 is

$$0 < |V_{HS}| < v_{off} \cdot \Big\{1 + \frac{(i+j) \cdot R_w}{R_{ON}}\Big\}, \tag{7a}$$

$$V_0 - v_{off} \cdot \Big\{1 + \frac{(i+j) \cdot R_w}{R_{ON}}\Big\} < |V_{VS}| < |v_{on}| \cdot \Big\{1 + \frac{(i+j-n) \cdot R_w}{R_{OFF}}\Big\}. \tag{7b}$$

Here, the output memristors are located in the column (row) j. Fig. 19 shows the upper bound of V_{HS} and upper and lower bounds of V_{VS} for different array sizes. Note that the lower bound of V_{HS} is always zero.

Even though the wire resistance R_w is negligible as compared to the minimum resistance of the memristor ($R_{ON} = 1k\Omega$), it plays a crucial role for the memristors, within an array, which are quite far from the row/column decoders. For example, within an array of size 200×200, the memristor at the middle of the array would experience a parasitic resistance of $100 \times 10\Omega = 1k\Omega$, which is equivalent to R_{ON}. Thus, it is essential to consider the role of the resistance of the wires while executing logic functions within memristive MPUs, similar to the consideration of the wires in memory arrays [53].

IV. Executing Logic Functions - A Full Adder Case Study

In this section, design algorithms for MAGIC within memristive MPUs are presented using an example of a one-bit full adder. The full adder consists of MAGIC NOR gates (and MAGIC NOT, which is a special case of MAGIC NOR) since NOR suffices as a complete logic structure. Assume the inputs of the full adder are *A*, *B*, and *C*, which are present inside the memory prior to computation. The results of the computation are sum S and carry C_{out}. The expression of C_{out} purely in the form of NOR operation of inputs can be expressed as

$$C_{out} = ((A+B)' + (B+C)' + (C+A)')'. \tag{8}$$

Similarly, S in the form of NOR and NOT of inputs and C_{out} (which are generated during the execution) can be expressed as

$$S = [[(A' + B' + C')' + \{(A+B+C)' + C_{out}\}']']'. \tag{9}$$

To execute the one-bit full adder within memristive MPU, (8) and (9) are employed using MAGIC NOR and NOT operations. The following subsections talk about optimized algorithms of an adder design.

A. An Adder Design within a Conventional Memory Crossbar

In conventional memory crossbar, the application of excitation voltages is enabled only from one direction (Sec. II-B). This limits the operation of MAGIC NOR to a particular row of the memory, where all inputs and outputs reside in the same row. Fig. 20 shows a row of conventional memory, over which, the adder is executed, utilizing nine memristors including inputs, outputs, and additional memristors to store intermediate results (*functional memristors*). For simplicity, the inputs *A*, *B*, *C* are assumed to be situated adjacent to one another as shown in Fig. 20.

Selecting the required steps to execute any logical operation depends on the latency and area constraints of the application. For example, in an application with latency optimization as the primary criterion, more memristors can be initialized simultaneously during the initial step. This approach eliminates the need for multiple intermediate initializations and thus lowers latency, while increasing the utilized area. When area optimization is the primary concern, fewer functional memristors are used and initialized multiple times during execution. This approach increases the latency (due to intermediate initializations), while decreasing area. The complete algorithm to evaluate a MAGIC adder on conventional memory is presented in the supplementary material (Tables SP1 and SP2). While optimizing area, the number of functional memristors utilized is four and the number of execution cycles is 15. For latency optimization, six additional functional memristors are used to reduce the execution time to 13 clock cycles.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

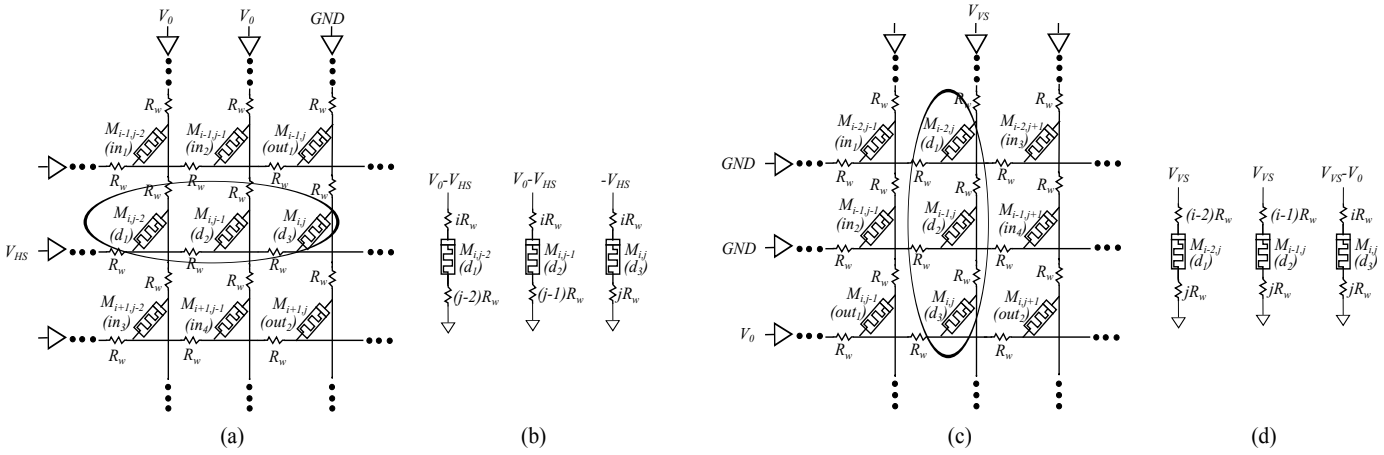

Fig. 18. Isolation of unselected cells with non-ideal wires. (a) row isolation and (b) its equivalent circuit, and (c) column isolation and (d) its equivalent circuit. The wires have a unit resistance of R_w.

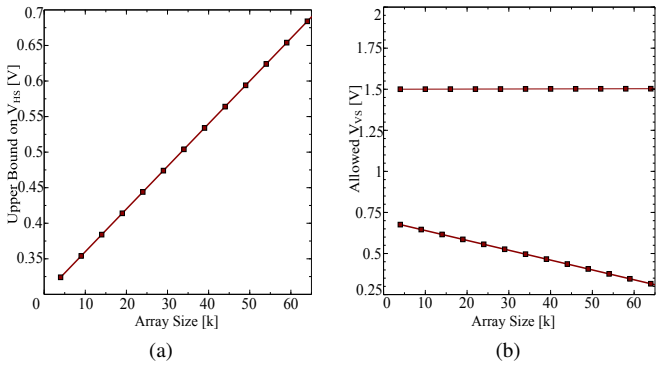

Fig. 19. (a) Upper bound of V_{HS} and (b) upper and lower bounds of V_{VS} for different values of $k \times k$. Unit resistance of non ideal wires is $R_w = 10\Omega$ and V_0 = 1V.

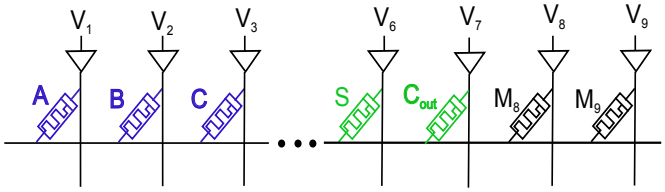

Fig. 20. Segment of a row of conventional memory crossbar over which a MAGIC adder is being implemented. A, B, C are the input memristors and M_4, ..., M_8 represent functional memristors (excluding M_7). The outputs C_{out} and S are generated, respectively, at M_7 and M_9.

B. An Adder Design within a Transpose Memory Crossbar

Executing logic within memristive MPUs of transpose memory gives more flexibility for the computation as operations are executed in both rows and columns. The added flexibility is more attractive for the computation of complex functions. In this simple case study of a full adder, we propose two algorithms to benefit from the capabilities of a transpose memory crossbar. These approaches distribute the intermediate data in an efficient manner such that the execution can be done over multiple rows/columns simultaneously, exploiting parallelism within the transpose memory. These approaches also ensure minimal overhead and reduce the amount of duplication of data.

1) Scheme-1 of MAGIC Adder Implementation: The first approach relies on organizing the data in an efficient way prior to computation and then executing multiple operations simultaneously. This can be achieved in two ways: a) by duplicating the data during initial write cycles or b) by copying and arranging the data during execution cycles. We prefer the latter approach as the former requires modification in data write pattern. Fig. 21 shows a 4×7 transpose memory crossbar utilized for the addition operation using this scheme. Assume that the inputs are stored within the same column: A at $M_{1,1}$, B at $M_{2,1}$, and C at $M_{3,1}$. The first step to compute an addition is to initialize (*i.e.*, write R_{ON}) at all functional and output memristors. Initialization is performed simultaneously for multiple memristors whenever it is possible. To enhance the computation, the inputs are duplicated to the next column such that $\widetilde{B}$ is stored next to *A*, $\widetilde{C}$ next to *B*, and $\widetilde{A}$ next to *C*, where $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$ represent the copied data and *A, B, C* represent the original data. After the copy operations, MAGIC voltages are applied as listed in the supplementary material (Table SP3). The table also reveals the equivalent logic operations performed in each cycle.

This approach utilizes 19 functional memristors and 16 computational steps (cycles) for execution. Nine cycles (cycles 3 to 11 in Table SP3) are required to copy data to the appropriate location and can be eliminated if data is duplicated in the appropriate locations when being stored. While duplicating data reduces the capacity of the memory, it requires only seven computational steps, making this approach faster than any other approach. The speed benefits from this approach are due to the execution of three NOR operations simultaneously (steps 12 and 13 in SP3).

2) Scheme-2 of MAGIC Adder Implementation: The necessity to copy data in Scheme-1 reduces the benefits from transpose memories. To remove the additional copy cycles from Scheme-1, assume all inputs are located within the same row: A at $M_{1,1}$, B at $M_{1,2}$, and C at $M_{1,3}$. After initialization of the functional memristors, $(A+B)'$, $(B+C)'$, and $(C+A)'$ are evaluated sequentially (rather than simultaneously as in Scheme-1). Fig. 22 shows a 4×9 transpose memory crossbar

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

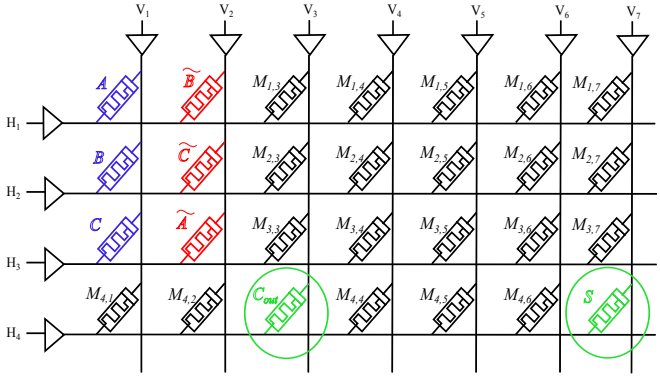

Fig. 21. Transpose memory crossbar utilized for executing an adder for Scheme-1. A, B and C are the inputs, $\widetilde{A}$, $\widetilde{B}$ and $\widetilde{C}$ are the copied data and C_{out} and S are, respectively, carry and sum outputs. Other memristors are functional memristors.

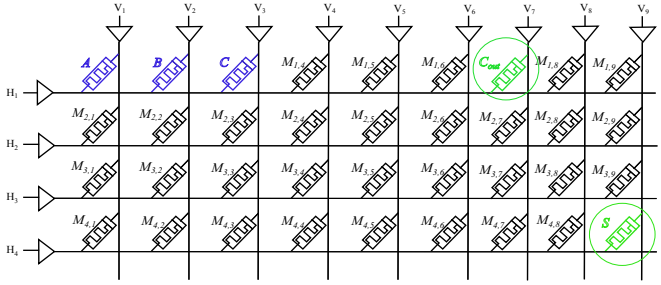

Fig. 22. Transpose memory crossbar utilized for executing an adder for Scheme-2. A, B and C are the inputs and C_{out} and S are, respectively, carry and sum outputs. Other memristors are functional memristors.

utilized for an addition operation and the sequence of operations is listed in the supplementary material (Table SP4). This approach requires 13 computational steps and ten functional memristors. Even though Scheme-2 reduces the number of parallel computations as compared to Scheme-1, it proves to be faster. Although for the case of an adder, Scheme-2 seems to be better than Scheme-1, Scheme-1 depicts how a transpose memory crossbar can be utilized for parallel operations over rows and columns and can be used for more complicated functions to add flexibility.

V. Evaluation and Comparison of Different Memristive Logic Families

In this section, we compare the proposed design techniques with previously proposed memristive stateful logic families (*i.e.*, IMPLY). Note that the comparison presented here is only among the pure logic families within memristive MPUs. We do not compare designs with conventional load-store based CMOS implementations since comparing von Neumann with non-von Neumann architecture is beyond the scope of this paper. Specifically, memristive logic families, such as Memristive Threshold Logic [54], CRS-based logic [55], [52], MRL [7], and stateful-NOR based reconfigurable architecture [56] are not considered since they are not pure logic within memristive MPUs, as not all of the logical states are represented as resistance. Additionally, CRS-based logic involves reading and sensing the intermediate data during execution. As a case

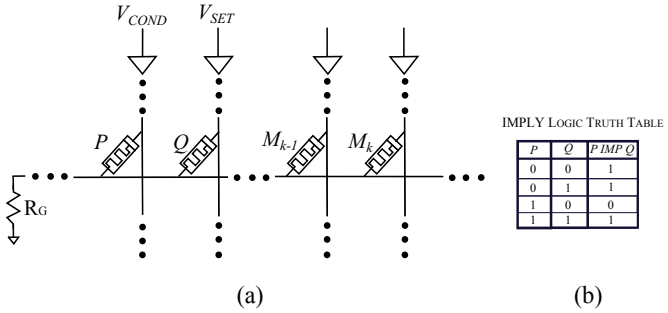

Fig. 23. IMPLY logic gate (a) within a memristive crossbar memory. The initial states of memristors P and Q are the inputs of the logic gate and output is the final state of memristor Q after applying voltages V_{COND} and V_{SET}. The load resistor R_G is connected at the common point of both the resistors. (b) IMPLY truth table.

study, an N-bit full adders are designed and evaluated for MAGIC and IMPLY logic families. Seven different designs are considered: a general IMPLY algorithm [19], serial and parallel IMPLY approaches [8], and the four proposed MAGIC NOR schemes. Area, speed, and energy of all six designs are evaluated and compared. To show the advantage of our approach, we also compare MAGIC and IMPLY-based logic execution on ISCAS-85 benchmark combinational circuits [57].

A. IMPLY-Based General Algorithm

IMPLY, also known as material implication, is a stateful logic family with two input memristors, P and Q, where one of the input memristors (Q) is also an output memristor. Fig. 23 shows the schematic and truth table of IMPLY within memristive MPU. To execute IMPLY, two voltages V_{COND} and V_{SET} ($V_{COND} < V_{SET}$) are applied. A resistor R_G is added to each row of the crossbar array and the logical IMPLY operation is achieved based on the ratio between P, Q, and R_G. Lehtonen *et al.* [19] have showed that any general Boolean function $f : B^n \longrightarrow B$ can be executed purely in terms of IMPLY and FALSE (a logic function that always yields logical zero as an output), using $n + 3$ memristors. While the algorithm is efficient in terms of area, it is not attractive in terms of latency as it requires $O(2^{kn})$ steps (where n is the number of inputs and k is the number of functional memristors).

B. IMPLY-Based Serial and Parallel Approaches

To improve the speed of IMPLY-based logic, Kvatinsky *et al.* proposed two techniques [8] - serial and parallel approaches. The serial approach relies on executing a single operation per clock cycle (*i.e.*, either IMPLY or FALSE), in which, all the memristors are located in a single row. In the parallel approach, multiple operations are executed per clock cycle, further reducing the overall latency of logical functions. A parallel approach requires connecting multiple rows of the memory crossbar, thus the crossbar structure is modified by adding switches to short different rows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

TABLE V
ENERGIES FOR IMPLY AND MAGIC NOR GATES FOR DIFFERENT INPUT COMBINATIONS

Input	IMPLY [fJ]	MAGIC NOR [fJ]
00	102.2	7.73
01	866.8	81.6
10	489.9	81.6
11	886.4	35.73

To execute a full adder with these approaches, sum and carry are computed as

$$S_k = (A_k \oplus B_k) \oplus C_k, \tag{10}$$

$$C_{out,k} = (A_k \rightarrow (B_k \rightarrow 0) \rightarrow ((C_k \rightarrow ((A_k \oplus B_k) \rightarrow 0)) \rightarrow 0). \tag{11}$$

where S_k and $C_{out,k}$ represent, respectively, the sum and carry out at the m^{th} stage of addition. To realize an XOR, two functional memristors are required. Complete S_k computation requires 26 computation steps. Carry computation requires three functional memristors.

C. Comparing Combinational Logic Designs within Memristive MPUs

To compare different memristive stateful logic families within memory, an N-bit addition is used as a case study. The execution of a full adder using MAGIC is carried out using the algorithms described in Section IV for each single bit addition and is extended for an N-bit ripple carry addition. In this subsection, the different approaches are compared in terms of latency, area, and energy. Latency is determined as the required number of clock cycles for an N-bit addition. Area is measured as the total number of utilized memristors (including inputs, outputs, and functional memristors) within the crossbar.

To evaluate the energy of addition, the energies of individual gates, IMPLY and MAGIC NOR, are evaluated using SPICE simulation for all input combinations. Then, the energy is averaged over all the input combinations using the measured gate energies to compute the energy for N-bit addition. The energies for IMPLY and MAGIC logic gates are listed in Table V. The evaluation is carried out to have an identical gate delay ($1.3ns$) for both the logic families. The circuit parameters chosen are $V_{SET} = 2V$, $V_{COND} = 1.5V$, and $R_G = 5k\Omega$ for IMPLY (Fig. 23) and $V_0 = 1V$ for MAGIC NOR (Fig. 6). Note that these energies do not incorporate initialization and FALSE operations, which are SET and RESET operations. The energy for SET and RESET operations is, respectively, 219.7 fJ ($V_{SET} = 2V$) and 34.26 fJ ($V_{RESET} = 1V$). IMPLY energy is higher than MAGIC energy for all approaches. IMPLY requires higher voltage levels to achieve the same delay time, which results in more current flowing through a relatively low resistance (*i.e.,* R_G).

The comparison between latency and area is listed in Table VI. The clock frequency of execution, f_{CLK}, is 0.77GHz (Section III-B), and area listed in the table incorporates only the functional memristors, since the number of input $(2N+1)$ and output $(N+1)$ memristors are identical for all of the different approaches. For an eight bit adder, the general algorithm requires 712 computational steps, the serial approach takes 232 steps, whereas the parallel approach utilizes 58 steps. The proposed MAGIC approaches in conventional and transpose memories with Scheme-1 and Scheme-2 utilize, respectively, 113, 121, and 83 execution steps to compute the result. Hence, MAGIC adder within memory is faster than IMPLY, unless the structure of the crossbar array is breached. Thus, the proposed MAGIC designs show improvement in the computational time and area as compared to IMPLY for conventional memristive crossbar memory. MAGIC within conventional memory crossbar is 2.05X faster (for $N = 8$) and 2.4X faster (on average for $N = 1$ to 1000) as compared to IMPLY (serial approach).

TABLE VI
COMPARISON OF MEMRISTIVE STATEFUL LOGIC FAMILIES FOR N-BIT ADDITION IN TERMS OF LATENCY AND AREA ($f_{CLK} = 0.77$ GHz)

Method of Execution	Latency (Cycles)	Area (# Memristors)
IMPLY base [19]	$89N$	4
IMPLY Serial [8]	$29N$	2
IMPLY Parallel [8]	$5N + 18$	$6N - 1$
MAGIC Conv. Area Optimized (this work)	$15N$	5
MAGIC Conv. Latency Optimized (this work)	$12N + 1$	$11N - 1$
MAGIC Trans. I (this work)	$15N + 1$	$22N - 3$
MAGIC Trans. II (this work)	$10N + 3$	$13N - 3$

Fig. 24 shows the latency for various lengths of addition. All of the proposed approaches are faster than IMPLY base (general algorithm) and serial approaches, while they are slower as compared to IMPLY parallel approach. The best among the proposed MAGIC approaches is Scheme-2 of transpose memory, which shows an average of 88% and 65% improvement for $N = 1$ to 1000 as compared to, respectively, IMPLY base and serial approaches. Fig. 25 shows the area (number of utilized memristors) for various lengths of addition. Out of the proposed MAGIC approaches, area optimized MAGIC adder within conventional memory has the best area efficiency. The additional functional memristors (five versus two in IMPLY serial approach) consume merely 0.62% more area on average for N = 1 to 1000 as compared to IMPLY serial approach. Furthermore, when the periphery area is considered, the proposed MAGIC within conventional memory becomes more attractive than the IMPLY serial approach since it does not use additional resistors and switches.

Fig. 26 shows the energy for various lengths of addition. Due to the lower energy requirement of MAGIC NOR gate, MAGIC-based approaches dominate this comparison. The energies for both MAGIC-based area and latency optimized

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

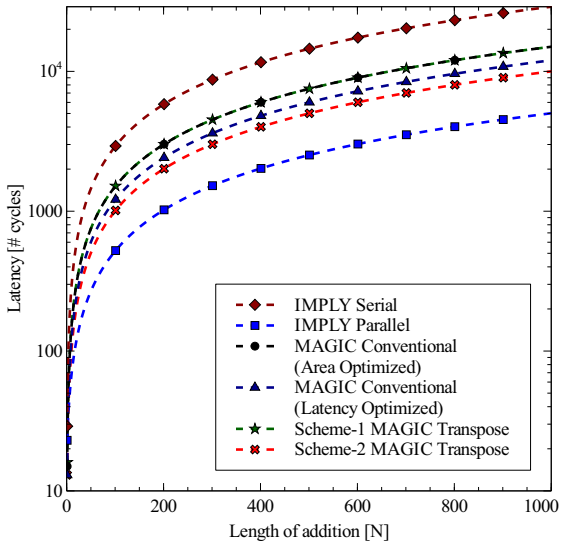

Fig. 24. Execution time vs. length of addition for different memristive stateful logic approaches.

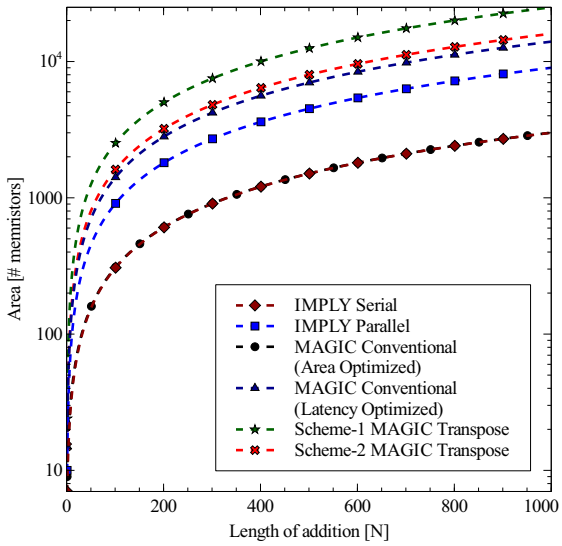

Fig. 25. Area vs. length of addition for different memristive stateful logic approaches. Area is measured by the number of memristors participating in the computation, peripheral circuitry are not included in the evaluation.

approaches on a conventional memory crossbar have the same energy consumption, since the same sequence of operation is being executed in both techniques, with a small change in the order of operation. MAGIC adder within conventional memory crossbar consumes 33.7% of total energy consumed in IMPLY (serial) approach.

Note that the half-select and isolation energies are not incorporated in these graphs. To make a generalized comparison of techniques, the energy consumption, including half-select and isolation energies, is shown in Fig. 27 for different values of length of addition (N) and array size. As the array size increases, there are more unselected and isolated cells, thus the energy consumption increases. Only in the case of the IMPLY parallel approach, due to the parallelism of FALSE

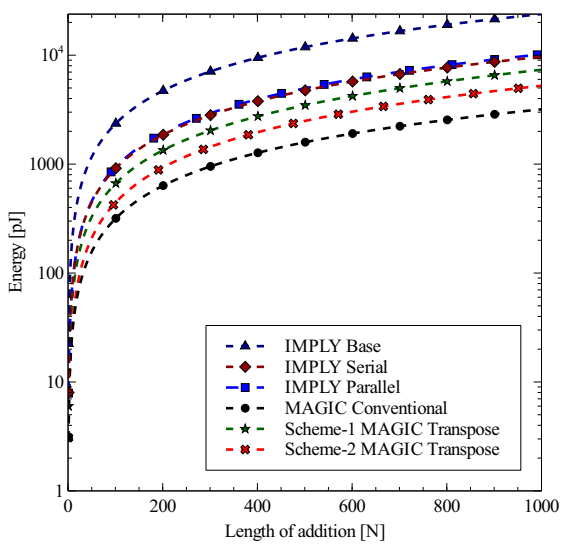

Fig. 26. Energy vs. length of addition for different memristive stateful logic approaches. Note that the energy consumption for both MAGIC optimizations within conventional memory are identical and denoted as 'MAGIC conventional.'

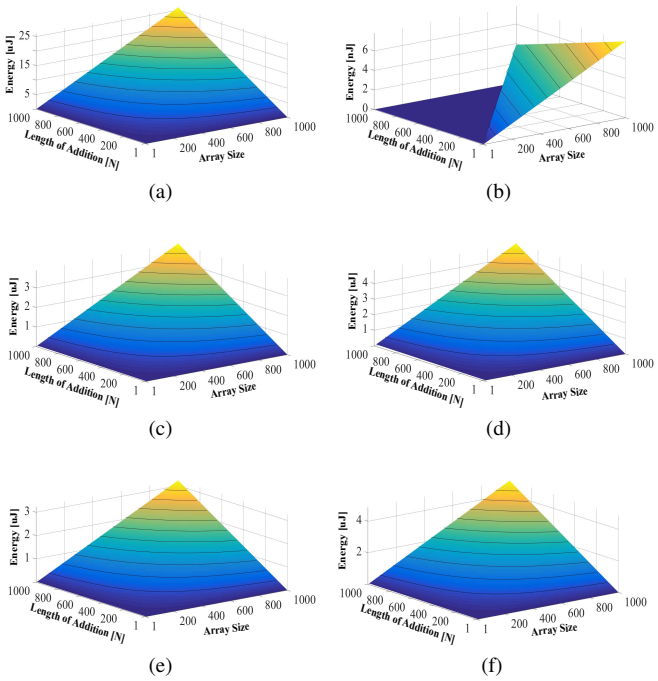

Fig. 27. Energy consumption including half-select energies for different values of length of addition (N) and array sizes for (a) IMPLY series, (b) IMPLY parallel, (c) MAGIC conventional (area optimized), (d) MAGIC conventional (latency optimized), (e) Scheme-1 of MAGIC transpose, and (f) Scheme-2 of MAGIC transpose approach.

and IMPLY operations, lower energy is consumed (in nJ) as compared to other approaches.

While the comparison in Table VI shows that the IMPLY-based parallel approach is advantageous over MAGIC in terms of latency and area, the MAGIC-based execution is generally more attractive than IMPLY for the following reasons:

- Conventional IMPLY (either base or series) [8], [19]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

TABLE VII
COMPARISON OF MAGIC AND IMPLY-BASED COMPUTING ON ISCAS-85 BENCHMARKS ($f_{CLK} = 0.77GHz$) [57]

Benchmark	MAGIC-based implementation			IMPLY-based implementation		
	# Cycles (WC)	# Memristors	Energy (pJ)	#Cycles (WC)	# Memristors	Energy (pJ)
$c432$	102	757	24.6	221	657	291.4
$c499$	66	1096	37.8	143	830	368.1
$c880$	144	1610	48.2	312	1438	603.3
$c1355$	144	2960	100.2	312	2486	1118.7
$c1908$	240	3620	118.8	520	3181	1254.1
$c2670$	192	4763	149.5	416	4111	1596
$c3450$	282	6499	200.2	611	5679	2220.4
$c5315$	294	9352	291	637	7993	3150.3
$c6288$	744	7728	151.3	1612	9600	4212.2
$c7552$	258	14540	461.2	559	12546	4983.8

requires modification in memory array structure due to the necessity to add a resistor and two different execution voltages (V_{COND} and V_{SET}) for execution. MAGIC-based execution does not require any additional devices to the memory structure and is supported by only a single execution voltage (V_0) [20].

- IMPLY parallel approach [8] requires significant modifications in the memory structure including additional switches and connections among different rows of the memory, which is not required for MAGIC execution.
- When comparing standard memory arrays (with the required additional resistors for IMPLY serial/base), MAGIC is 2.4X faster than IMPLY serial for the adder case study. IMPLY outperforms MAGIC only when they are evaluated in the modified arrays (parallel IMPLY versus transpose memory).
- MAGIC execution dedicates a separate memristor for storing output, thus, all the inputs are preserved, as opposed to IMPLY, where one of the input memristors acts as an output memristor. Hence, MAGIC execution allows non-destructive operation and is therefore more appropriate for logic execution within MPU.
- Complex functions are implemented using NOR in the case of MAGIC execution and IMPLY and FALSE in the case of an IMPLY approach. NOR operation is more intuitive as compared to IMPLY and FALSE, and thus, design automation can be supported using standard automation tools.

The proposed MAGIC design is also compared with IMPLY on ISCAS-85 benchmark circuits [57], in terms of worst case (WC) number of execution cycles (for frequency $f_{CLK} = 0.77GHz$), number of utilized memristors, and energy. For each benchmark, all the basic gates are designed using the basis functions in respective approaches (MAGIC and IMPLY). To compute overall latency of a circuit, the worst path delays of all the levels are added. Area and energy calculation is carried out by finding the area and energy of basis functions in each approach, extending them to basic gates from the benchmark, and calculating them for all the required gates in the circuit. The comparison shows that MAGIC-based computing is the clear conqueror against IMPLY in speed (at least 2X faster) and energy (at least 10X improvement), with little overhead in terms of area for all tested benchmarks.

VI. CONCLUSIONS

Logic within memristive MPUs is an attractive approach to enable novel non-von Neumann architectures, where pure computation is performed within the memory. These architectures significantly reduce data transfer and therefore can lead to extremely energy efficient computers in the future.

We demonstrate the use of MAGIC gates within memristive MPUs. Different design issues, such as determining circuit parameters and the isolation of unselected cells are considered to allow proper operation and efficient designs. The proposed techniques are evaluated and compared with previously proposed approaches that are based on an IMPLY logic gate.

While comparing MAGIC and IMPLY within a standard memory crossbar structure, MAGIC outperforms IMPLY both in speed and energy. The proposed transpose memory, that gives additional functionality to the memristive crossbar, has similar performance as well as energy as compared to MAGIC within conventional memory for the case study of a full adder. While it seems that the benefits from a transpose memory are limited, the flexibility gained by this memory structure can be exploited for more complex functions that can benefit from more parallelism, resulting in better speed and energy.

ACKNOWLEDGMENT

The authors would like to thank Mr. Pushparaj Paradkar (Microelectronics Lab, BITS Pilani, K.K. Birla Goa Campus) for offering his excellent technical support and Prof. Mark Horowitz (Stanford University) for his support and useful remarks. Furthermore, the authors would also like to thank anonymous reviewers for improving the quality of this manuscript by their useful comments.

REFERENCES

[1] L. Chua, "Memristor-The missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, pp. 507–519, Sep 1971.
[2] L. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, pp. 209–223, Feb 1976.
[3] Y. Ho, G. Huang, and P. Li, "Nonvolatile memristor memory: Device characteristics and design implications," in *Proc. IEEE Int. Conf. Comput.-Aided Design - Dig. Tech. Papers*, pp. 485–490, Nov 2009.

[4] Q. Xia et al., "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic," *Nano Lett.*, vol. 9, no. 10, pp. 3640–3645, 2009.

[5] J. Borghetti et al., "Memristive Switches Enable Stateful Logic Operations via Material Implication," *Nature*, vol. 464, pp. 873–876, Apr. 2010.

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

[7] S. Kvatinsky et al., "MRL - Memristor Ratioed Logic," in *Int. Workshop Cellular Nanoscale Networks Applicat.*, pp. 29–31, Aug. 2012.

[8] S. Kvatinsky et al., "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, pp. 2054–2066, Oct 2014.

[9] S. H. Jo et al., "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[10] B. Linares-Barranco et al., "On Spike-Timing-Dependent-Plasticity, Memristive Devices, and building a Self-Learning Visual Cortex," *Frontiers Neuroscience*, vol. 5, no. 26, 2011.

[11] D. Soudry et al., "Memristor-Based Multilayer Neural Networks with Online Gradient Descent Training," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 26, no. 10, pp. 2408–2421, 2015.

[12] M. Horowitz, "1.1 Computing's Energy Problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 10–14, Feb 2014.

[13] E. Linn et al., "Beyond von Neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, pp. 305205:1–6, July 2012.

[14] M. Ziegler and M. Stan, "CMOS/nano co-design for crossbar-based molecular electronic systems," *IEEE Trans. Nanotechnol.*, vol. 2, pp. 217–230, Dec 2003.

[15] A. Dehon, "Nanowire-based Programmable Architectures," *J. Emerging Technol. Comput. Syst.*, vol. 1, pp. 109–162, July 2005.

[16] K. Likharev and D. Strukov, "CMOL: Devices, Circuits, and Architectures," in *Introducing Molecular Electron.* (G. Cuniberti, K. Richter, and G. Fagas, eds.), vol. 680 of *Lecture Notes Physics*, pp. 447–477, Springer Berlin Heidelberg, 2005.

[17] P. Mane et al., "Implementation of NOR logic based on material implication on CMOL FPGA architecture," in *Proc. IEEE Conf. VLSI Design*, pp. 523–528, Jan 2015.

[18] S. Paul and S. Bhunia, "A scalable memory-based reconfigurable computing framework for nanoscale crossbar," *IEEE Trans. Nanotechnol.*, vol. 11, pp. 451–462, May 2012.

[19] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanoscale Architectures NANOARCH '09*, pp. 33–36, July 2009.

[20] S. Kvatinsky et al., "MAGIC- Memristor - Aided loGIC," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, pp. 895–899, Nov 2014.

[21] H. Li et al., "Write disturb analyses on half-selected cells of cross-point rram arrays," in *Proc. IEEE Int. Rel. Physics Symp.*, pp. MY.3.1–MY.3.4, June 2014.

[22] J. Thatcher et al., "NAND flash solid state storage for the enterprise, an in-depth look at reliability," in *Proc. Solid State Storage Initiative (SSSI), 2009 © Storage Network Ind. Assoc.*

[23] E. Ou and S. Wong, "Array architecture for a nonvolatile 3-dimensional cross-point resistance-change memory," *IEEE J. Solid-State Circuits*, vol. 46, pp. 2158–2170, Sept 2011.

[24] L. Chua, "Resistance switching memories are memristors," *Appl. Physics A*, vol. 102, pp. 765–783, Jan. 2011.

[25] J. Borghetti et al., "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proc. Nat. Academy Sci.*, vol. 106, no. 6, pp. 1699–1703, 2009.

[26] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, pp. 485203:1–7, Nov. 2011.

[27] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, pp. 13–24, Jan. 2013.

[28] J. J. Yang et al., "High switching endurance in TaOx memristive devices," *Appl. Physics Lett.*, vol. 97, no. 23, pp. 232102:1–3, 2010.

[29] J. Nickel, "Memristor material engineering: From flash memory replacement towards a universal memory," in *IEEE IEDM Adv. Memory Technol. Workshop*, pp. 142–147, Oct. 2011.

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

[31] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, vol. 18, pp. 210–214, June 2009.

[32] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A Versatile Memristor Model With Nonlinear Dopant Kinetics," *IEEE Trans. Electron Devices*, vol. 58, pp. 3099–3105, Sept 2011.

[33] J. J. Yang et al., "Memristive Switching Mechanism for Metal/Oxide/Metal Nanodevices," *Nature Nanotechnology*, vol. 3, pp. 429–433, July 2008.

[34] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *Int. Workshop Cellular Nanoscale Networks Applicat.*, pp. 1–4, Feb 2010.

[35] M. D. Pickett et al., "Switching dynamics in titanium dioxide memristive devices," *J. Appl. Physics*, vol. 106, pp. 074508:1–6, Oct. 2009.

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

[37] S. Kvatinsky, E. Friedman, A. Kolodny, and U. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, pp. 211–221, Jan 2013.

[38] S. Kvatinsky, M. Ramadan, E. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, pp. 786–790, Aug. 2015.

[39] E. Yalon et al., "Resistive switching in HfO_2 probed by a metal-insulator-semiconductor bipolar transistor," *IEEE Electron Device Lett.*, vol. 33, pp. 11–13, Jan 2012.

[40] A. Chanthbouala et al., "A ferroelectric memristor," *Nature Materials*, vol. 11, pp. 860–864, Oct 2012.

[41] C. Yakopcic et al., "A memristor device model," *IEEE Electron Device Lett.*, vol. 32, pp. 1436–1438, Oct 2011.

[42] Y.-C. Chen et al., "An access-transistor-free (0T/1R) non-volatile resistance random access memory (RRAM) using a novel threshold switching, self-rectifying chalcogenide device," in *IEEE Int. IEDM '03 Tech. Dig. Electron Devices Meeting*, pp. 37.4.1–37.4.4, Dec 2003.

[43] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, pp. 176 – 183, Feb. 2013.

[44] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, pp. 156–160, July 2013.

[45] W. Lynch, "Worst-case analysis of a resistor memory matrix," *IEEE Trans. Comput.*, vol. C-18, pp. 940–942, Oct. 1969.

[46] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proc. IEEE*, vol. 100, pp. 2021–2032, June 2012.

[47] C.-M. Jung, J.-M. Choi, and K.-S. Min, "Two-Step Write Scheme for Reducing Sneak-Path Leakage in Complementary Memristor Array," *IEEE Trans. Nanotechnol.*, vol. 11, pp. 611–618, May 2012.

[48] C. Xu, X. Dong, N. Jouppi, and Y. Xie, "Design implications of memristor-based rram cross-point structures," in *Design, Automation Test in Europe Conf. Exhibition (DATE), 2011*, pp. 1–6, March 2011.

[49] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar, "Resistive GP-SIMD processing-in-memory," *ACM Trans. Architecture Code Optimization.*, vol. 12, no. 4, p. 57, 2016.

[50] T.-y. Liu et al., "A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, pp. 210–211, Feb 2013.

[51] M. A. Zidan et al., "Memristor multiport readout: A closed-form solution for sneak paths," *IEEE Trans. Nanotechnol.*, vol. 13, pp. 274–282, Mar. 2014.

[52] E. Linn, R. Rosezin, C. Kgeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Materials*, vol. 9, pp. 403–406, May 2010.

[53] J. Liang, S. Yeh, S. S. Wong, and H.-S. P. Wong, "Effect of wordline/bitline scaling on the performance, energy consumption, and reliability of cross-point memory array," *J. Emerg. Technol. Comput. Syst.*, vol. 9, pp. 9:1–9:14, Feb. 2013.

[54] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "An energy-efficient memristive threshold logic circuit," *IEEE Trans. Comput.*, vol. 61, pp. 474–487, Apr. 2012.

[55] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 5, pp. 64–74, Mar. 2015.

[56] P. Mane et al., "Stateful-NOR based reconfigurable architecture for logic implementation," *Microelectronics Journal*, vol. 46, pp. 551 – 562, June 2015.

[57] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *Design Test of Comput., IEEE*, vol. 16, no. 3, pp. 72–80, 1999.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNANO.2016.2570248, IEEE Transactions on Nanotechnology

Nishil Talati is a graduate student at the Andrew and Erna Viterbi Faculty of Electrical Engineering at the Technion Israel Institute of Technology, Haifa, Israel. He received the B.Sc. degree in Electrical Engineering from Birla Institute of Technology & Science (BITS) Pilani, K.K. Birla Goa Campus, Goa, India in 2016. He worked as a research scholar at the department of Computer Science and Engineering at the University of Michigan, Ann Arbor, USA during Fall, 2015. His research interests include novel computer architecture and circuit design using emerging memory technologies, specifically, Resistive RAM (RRAM).

Saransh Gupta is an undergraduate student at the department of Electrical, Electronics, & Instrumentation Engineering at Birla Institute of Technology & Science (BITS) Pilani, K.K. Birla Goa Campus, Goa, India. His research interests include novel VLSI architectures and circuit design.

Pravin Mane is pursuing his Ph.D. at the department of Electrical, Electronics, & Instrumentation, at Birla Institute of Technology & Science (BITS) Pilani, K.K. Birla Goa Campus, Goa, India. He received B.Sc. and M.Sc. in Electronics Engineering from Shivaji University Kolhapur, Maharashtra, India in 1998 and from Indian Institute of Technology (IIT), Roorkee, Uttarakhand, India in 2006, respectively. He has worked as faculty in Mody Institute of Technology & Science, Lakshmangarh, Rajsthan and Vidyalankar Institute of Technology, Mumbai, India. He has been working as a lecturer at BITS Pilani at Goa, India since 2009. His research interests include reconfigurable architectures and FPGA based designs.

Shahar Kvatinsky is an assistant professor at the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion Israel Institute of Technology. He received the B.Sc. degree in computer engineering and applied physics and an MBA degree in 2009 and 2010, respectively, both from the Hebrew University of Jerusalem, and the Ph.D. degree in electrical engineering from the Technion Israel Institute of Technology in 2014. From 2006 to 2009 he was with Intel as a circuit designer and was a post-doctoral research fellow at Stanford University from 2014 to 2015. Kvatinsky is an editor in Microelectronics Journal and has been the recipient of the 2015 IEEE Guillemin-Cauer Best Paper Award, 2015 Best Paper of Computer Architecture Letters, Viterbi Fellowship, Jacobs Fellowship, the 2014 Hershel Rich Technion Innovation Award, 2013 Sanford Kaplan Prize for Creative Management in High Tech, 2010 Benin prize, and six Technion excellence teaching awards. His current research is focused on circuits and architectures with emerging memory technologies and design of energy efficient architectures.