Microelectronics Journal 45 (2014) 1429–1437

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Logic operations in memory using a memristive Akers array

Yifat Levy [a], Jehoshua Bruck [b], Yuval Cassuto [a], Eby G. Friedman [c], Avinoam Kolodny [a], Eitan Yaakobi [b], Shahar Kvatinsky [a,*]

[a] Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel
[b] Electrical Engineering, California Institute of Technology, Pasadena, CA 91125, USA
[c] Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627, USA

ARTICLE INFO

Article history:
Received 4 March 2014
Received in revised form
27 May 2014
Accepted 25 June 2014
Available online 13 August 2014

Keywords:
Memristor
Memristive systems
Logic array
Memory array
von Neumann Architecture
Akers logic array

ABSTRACT

In-memory computation is one of the most promising features of memristive memory arrays. In this paper, we propose an array architecture that supports in-memory computation based on a logic array first proposed in 1972 by Sheldon Akers. The Akers logic array satisfies this objective since this array can realize any Boolean function, including bit sorting. We present a hardware version of a modified Akers logic array, where the values stored within the array serve as primary inputs. The proposed logic array uses memristors, which are nonvolatile memory devices with noteworthy properties. An Akers logic array with memristors combines memory and logic operations, where the same array stores data and performs computation. This combination opens opportunities for novel non-von Neumann computer architectures, while reducing power and enhancing memory bandwidth.

1. Introduction

Conventional computers are based on a von Neumann architecture, where separate units process and store data. A different approach is to process data within the same unit that stores the data (i.e., process data within memory). An illustration of both architectures is shown in Fig. 1. In this paper, a hardware version of processing within memory is proposed. The proposed circuit is based on a study of rectangular logic arrays, first proposed in 1972 by Sheldon Akers [1].

In an *Akers logic array* (or, in short, an *Akers array*), the execution of any Boolean function is performed by flowing data across an array of primitive logic cells. The data are transferred from each primitive logic cell to neighboring cells, as shown in Fig. 2a. The operation of an Akers array is similar to systolic array [2] and cellular automata [19]. The primitive logic cell has three inputs and two outputs, as shown in Fig. 2b. The inputs of the primitive logic cell include two control inputs x and y and a variable input z, which is replaced in our circuit by an internal state (i.e., the stored data). The primitive logic cell performs a predefined logical operation $f(x, y, z)$, which is described below. The output of each primitive logic cell is used as control inputs x and y of, respectively, the bottom and right neighboring primitive logic cells.

To execute any Boolean function within an Akers array, specific input values are inserted as control inputs into the left-most column and the upper-most row. The control input y of the left-most column is set to 1 for all rows, and the control input x of the upper-most row is set to 0 for all columns, as shown in Fig. 2a. These control inputs along with the array structure and the function $f(x, y, z)$ determine the Boolean function computed by the array. The inputs to this Boolean function are the bits stored within the array cells. The output of the Boolean function computed by the Akers array is the output of the primitive logic cell at the bottom right of the array. It is also possible to define multiple Boolean functions (or, alternatively, a multi-bit output) on the same Akers array, in which case additional primitive cell outputs are used as external functional outputs. To date, an Akers array has been treated as a mathematical concept since the benefit of an Akers logic array with conventional semiconductor technology (i.e., CMOS technology) is limited, as described in Section 2.

The emergence of memristive technologies [3] enables the integration of computation and memory, including logic within memory [5–6,20–26]. The high density of memristors and compatibility with CMOS makes an Akers array with memristors practical.

* Corresponding author.
E-mail addresses: yifatl@tx.technion.ac.il (Y. Levy), bruck@caltech.edu (J. Bruck), ycassuto@ee.technion.ac.il (Y. Cassuto), friedman@ece.rochester.edu (E.G. Friedman), kolodny@ee.technion.ac.il (A. Kolodny), yaakobi@caltech.edu (E. Yaakobi), skva@tx.technion.ac.il (S. Kvatinsky).

http://dx.doi.org/10.1016/j.mejo.2014.06.006
0026-2692/

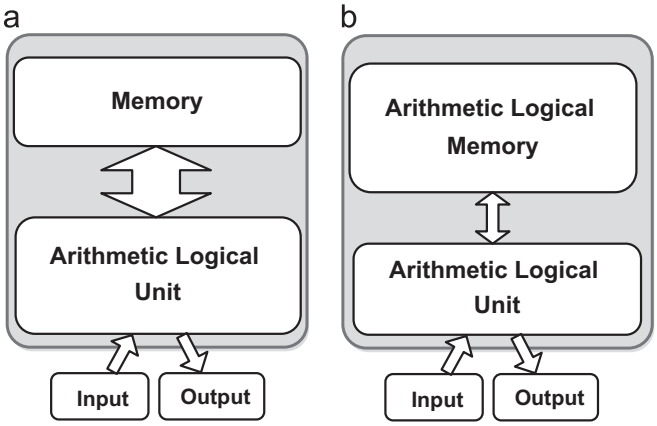

Fig. 1. Different computer architectures. (a) von Neumann architecture – separate memory and an ALU. (b) Processing within memory architecture (e.g., memristive Akers array). The memory can also process data. The size of the ALU is therefore smaller and the required memory bandwidth lower (schematically represented by the thickness of the arrow between the memory and the ALU).

In this paper, a memristive Akers array is proposed, where the variables z are stored within the memristive cells, and the control inputs x and y are voltages. The proposed memristive Akers array serves as a practical example of in-memory computation.

The design of the proposed memristive Akers array is demonstrated here by a small example of a four by four array, producing a variety of array operations, including a bit sorting algorithm as a case study. The rest of the paper is organized as follows. In Section 2, background describing both the Akers array and memristors is provided. The proposed memristive Akers array is described and evaluated in, respectively, Sections 3 and 4, followed by a discussion of design considerations for larger arrays in Section 5. A small example of different array operations is described in Section 6, followed by some concluding remarks in Section 7.

2. Background

In this section, the theory of the original Akers logic arrays is described and the basic principles of memristive devices are reviewed, including the model used in this paper for evaluating the proposed memristive Akers array.

2.1. Akers logic array

An Akers logic array is a two-dimensional array of identical primitive logic cells connected in a rectangular grid, as shown in Fig. 2a. The primitive logic cell in the array is a three input logic gate that executes the logical operation,

$$f(x,y,z)=x\bar{z}+yz. \quad (1)$$

Note that in the original Akers array [1], four alternative logical operations that generate the correct behavior of the array are proposed. In this paper, only Eq. (1) is used due to the easy implementation with memristors.

The output of each primitive logic cell is transferred to the two neighboring primitive logic cells in the array – one below and one to the right of the array. The transferred data are the x and y control inputs of, respectively, the vertical and horizontal neighbors, as shown in Fig. 2a. The control input y of the left-most column is set to 1 for all rows, and the control input x of the uppermost row is set to 0 for all columns.

The execution of a Boolean function is performed by organizing the contents of the array cells according to the particular specification, and reading the functional output from the output of the lower-right cell (or from multiple cell outputs in the case of a Boolean function with a multiple bit output or, alternatively, multiple Boolean functions simultaneously computed within the same array). Hence, the same array can be used for different Boolean functions, each specifying a different organization of inputs. Examples of several Boolean functions are illustrated in Fig. 3.

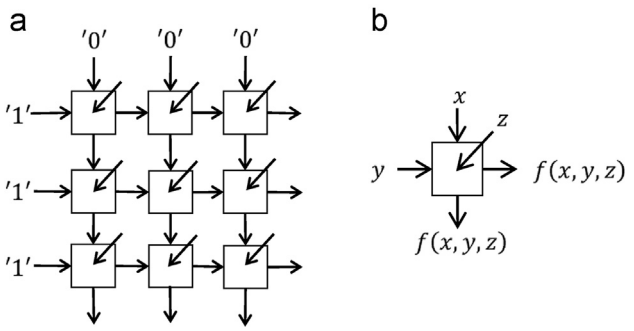

Fig. 2. Akers logic array. (a) An example of a three by three Akers array structure. (b) A primitive logic cell with three inputs x, y, z and two identical outputs $f(x, y, z)$.

Sorting of four bits $\{z_0, z_1, z_2, z_3\}$ is shown in Fig. 3a. The binary sorting function on n inputs is defined as the n Boolean functions $f_0, \ldots, f_{n-1}$, where $f_i(z_0, \ldots, z_{n-1}) = 1$ if the number of "1" inputs among $z_0, \ldots, z_{n-1}$ is greater than i (i.e., f_0 is the maximum value and f_{n-1} is the minimum of the output). For the sorting function, each input variable of the sorting Boolean function is replicated a number of times up to the number of inputs [1]. For example, z_3 is replicated four times, while z_1 is replicated two times. The number of primitive logic cells is therefore $\sum_{i=0}^{n-1}(i+1) = (n^2/2)+(n/2)$ - where n is the number of inputs to the sorting Boolean function. The output bits of the sorting Boolean function are placed along the diagonal of the array, as shown in Fig. 3a.

Another example for a Boolean function within an Akers array is a four-bit XOR [1], as shown in Fig. 3b. The variable inputs of the primitive logic cells are arranged similarly to the sorting array, where the complementary value of the XOR inputs are also stored as input variables of the primitive logic cells. The output of the XOR operation is the output of the bottom right primitive logic cell. The number of primitive logic cells for an n-bit XOR is n^2.

Since the inputs of the Boolean function must be replicated within an array, the number of primitive logic cells increases quadratically with the number of inputs of the Boolean function. A CMOS Akers logic array therefore requires significant area, making an Akers array impractical with standard CMOS. In contrast, the density and circuit architecture of memristive devices make the Akers array natural for memories. A memristive Akers array within memory can be denser than standard SRAM (without computation capabilities), as listed in Table 1.

2.2. Memristors

Memristors and memristive devices [3,7] are two-port passive elements with varying resistance. The change in the resistance of these devices depends on the current flowing through the device (or, alternatively, the voltage across the device), as shown in Fig. 4. While in theory the change in the resistance of a memristor depends directly on the current (or voltage), for memristive devices the dependence can be more complicated and described by internal state variables [7]. In this paper, the term *memristor* is used to describe both memristors and memristive devices.

Since 2008, numerous emerging nonvolatile memory technologies have been connected to the theory of memristors [8–12]. These technologies are nonvolatile, fast, dense, CMOS compatible, low power, and have high write endurance. The compatibility of memristors with

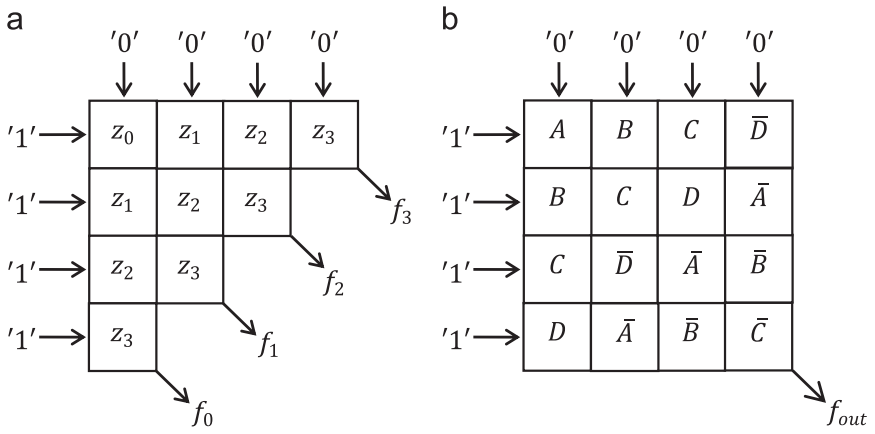

Fig. 3. Four-bit input structure for an Akers arrays for Boolean functions (a) Sort $\{z_0, z_1, z_2, z_3,\}$, and (b) $XOR(A, B, C, D)$.

Table 1
Area of memory technologies F – feature size, T – transistor, C – capacitor, and R – resistive device (memristor).

Technology	Memory cell	Area per cell (F^2)	Computing capabilities	Sequential
SRAM	6T	140	No	–
DRAM	1T 1C	6–12	No	–
Flash	1T	4	No	–
RRAM (memristive memory, single device per cell)	1R	4	Yes [5–6,25–26]	Yes
Complementary resistive switches	2R	4–8	Yes [24]	Yes
Akers array within memory	2R 4T	20–90	Yes	No

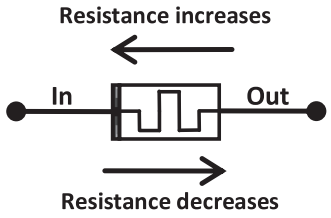

Fig. 4. Memristor symbol. The polarity of the memristor is represented by the thick black line. When current flows into the device, the resistance of the device increases. When current flows out of the device, the resistance of the device decreases.

CMOS enables the use of memristors not only as memory, but also as logic circuits [4–6,13,20–26].

Several models have been proposed to describe the behavior of memristors. In this paper, the TEAM model is used [14]. The TEAM model is general and can fit memristors from different technologies. In the TEAM model, it is assumed that a memristor has current thresholds, i_{off} and i_{on}, and an internal state variable x. When the current flowing through the memristor is above the current thresholds, the memristor changes state either from R_{on} to R_{off} or from R_{off} to R_{on} depending upon the original state and direction of the current. The voltage–current relationship and the change in state variable are described by

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

$$\frac{dx(t)}{dt} = \begin{cases} k_{off} \cdot \left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} \cdot f_{off}(x), & 0 < i_{off} < i, \\ k_{on} \cdot \left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} \cdot f_{on}(x), & i < i_{on} < 0, \\ 0, & otherwise, \end{cases} \tag{3}$$

where R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, x_{on} and x_{off} are, respectively the minimum and maximum value of the state variable x, $f_{on}(x)$ and $f_{off}(x)$ are window functions (the TEAM window function is used in this paper), and k_{off}, k_{on}, α_{off}, and α_{on} are fitting parameters. An example of an I–V curve of the TEAM model is shown in Fig. 5.

3. Proposed memristive Akers logic array

As previously mentioned, an Akers array with conventional CMOS technology is impractical due to the significant area requirements. The use of memristors, which are dense and fabricated physically above the CMOS transistors, significantly reduces the area.

The proposed memristive Akers primitive logic cell is based on the structure of complementary memristors (or complementary resistive switches, CRS) [15,16]. In the proposed memristive realization of an Akers array, the input variable z is the stored internal state of a memristor. The inputs of the executed Boolean function are therefore treated as stored data within a memristive memory array. In this section, the structure of the primitive logic cell is described as well as the operation of the array.

3.1. Primitive logic cell structure

The proposed primitive logic cell realizes the logical connectivity described by Eq. (1). The primitive cell consists of two anti-serial memristors (connected with opposite polarity), as shown in Fig. 6a. The control inputs of the primitive logic cell x and y are voltages (logical one and zero are, respectively, a positive voltage V_r and ground). The variable input z is the stored logical state of memristor M_Z, which is represented by the resistance of the device (low and high resistances are considered, respectively, as logical one and zero). The memristor $M_{\bar{Z}}$ has the complementary logical state of M_Z. The stored logical state of M_Z and $M_{\bar{Z}}$ are written during a write operation prior to execution.

Ideally, the memristors can be modeled as switches, where a high resistance is an open circuit and a low resistance is a short circuit, as shown in Fig. 6b. In an ideal model, one switch is open and the other switch is closed. If z is logical one, the switch of z is

closed and the logical value of y is transferred to the output. If z is logical zero, the switch is open and the complementary switch is closed, transferring x to the output.

The precise output of the primitive logic cell is the result of a voltage divider between M_Z and $M_{\bar{Z}}$. The output voltage V_f is

$$V_f = \frac{V_y - V_x}{R_Z + R_{\bar{Z}}} \cdot R_{\bar{Z}} + V_x, \tag{4}$$

where R_Z and $R_{\bar{Z}}$ are, respectively, the resistance of memristors M_Z and $M_{\bar{Z}}$, varying from R_{on} to R_{off}. V_x and V_y are the input voltages x and y. The output voltage V_f for different input conditions is listed in Table 2, demonstrating that, as required, the primitive logic cell indeed executes the Boolean function (1).

3.2. Logic array operation

The Akers logic array is an array of primitive logic cells that can also be used as a memory array, as shown in Fig. 7. Unlike regular memory arrays, the memristive Akers logic array can compute different Boolean functions in addition to storing data. The computation of Boolean functions within the logic array is divided into two stages. The initial stage is a "write" operation to the memristors. In this stage, the initial logical state of memristors M_Z and $M_{\bar{Z}}$ is simultaneously written. This stage can be part of a regular write operation of the memory or, alternatively, an explicit initialization prior to computing the Boolean function. In this paper, initialization of a single primitive logic cell is evaluated.

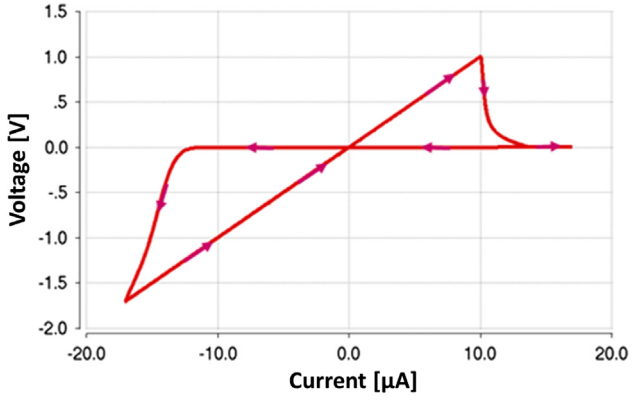

Fig. 5. Current–voltage characteristics of a memristor based on the TEAM model [14] for a sinusoidal current input with an amplitude of 17 μA and frequency of 100 kHz. The memristor parameters are listed in Table 3.

Writing to the array (e.g., addressing the specific primitive cells within the array and parallelizing the writes) is only briefly discussed since this process is similar in any CRS-based memory (e.g., see [16]). Relevant adjustments (e.g., adding CMOS selectors to achieve isolation between the primitive cells and maintain regular read and write operations), however, need to be performed to achieve a memory integrated with an Akers logic array, as shown in Fig. 7c.

The second stage executes the Boolean function. In this stage, a low voltage is used to ensure that the resistance of the memristors in the array does not change.

1) Stage 1 – initialization of the primitive logic cells (write)
   Initialization of the logical states of M_Z and $M_{\bar{Z}}$ is simultaneously achieved due to the anti-serial connection of both memristors. In the complementary structure, applying a sufficiently high voltage to both memristors switches both memristors to different resistances, where one memristor achieves a high resistance and the other memristor achieves a low resistance. The write procedure in a complementary pair of memristors is shown in Fig. 8.
   To write a logical one to M_Z, the resistances M_Z and $M_{\bar{Z}}$ are required to be, respectively, a low and high resistance. The write procedure therefore applies a sufficiently positive voltage V_w to y while grounding x. To write a logical zero to M_Z, the write procedure applies V_w to x while grounding y, or alternatively, apply $-V_w$ to y and grounding x. At the end of the write operation, the resistance of M_Z and $M_{\bar{Z}}$ are R_{ON} and R_{OFF}, where the resistance of one memristor is R_{ON} and the resistance of the other memristor is R_{OFF}.

Table 2
Output voltage of primitive logic cell from Eq. (4).

x	y	z	R_Z	V_f – derived from Eq. (4)	$f(x, y, z)$
0	0	0	R_{OFF}	$\mathbf{0}$	0
0	0	1	R_{ON}	$\mathbf{0}$	0
0	1	0	R_{OFF}	$\frac{R_{ON}}{R_{OFF}+R_{ON}} \cdot \boldsymbol{V_r}$	0
0	1	1	R_{ON}	$\frac{R_{OFF}}{R_{OFF}+R_{ON}} \cdot \boldsymbol{V_r}$	1
1	0	0	R_{OFF}	$\frac{R_{OFF}}{R_{OFF}+R_{ON}} \cdot \boldsymbol{V_r}$	1
1	0	1	R_{ON}	$\frac{R_{ON}}{R_{OFF}+R_{ON}} \cdot \boldsymbol{V_r}$	0
1	1	0	R_{OFF}	$\boldsymbol{V_r}$	1
1	1	1	R_{ON}	$\boldsymbol{V_r}$	1

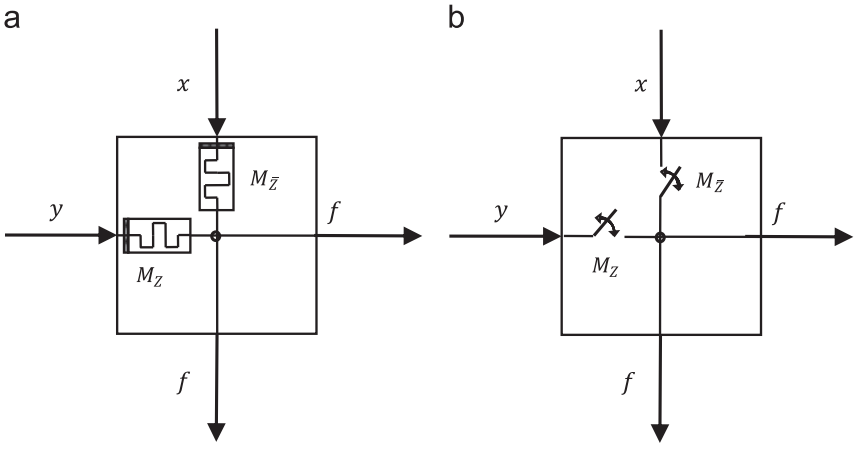

Fig. 6. Primitive logic cell. (a) The proposed primitive logic cell using memristors. (b) A behavioral model of the basic logic cell, where the memristors are modeled as ideal switches.

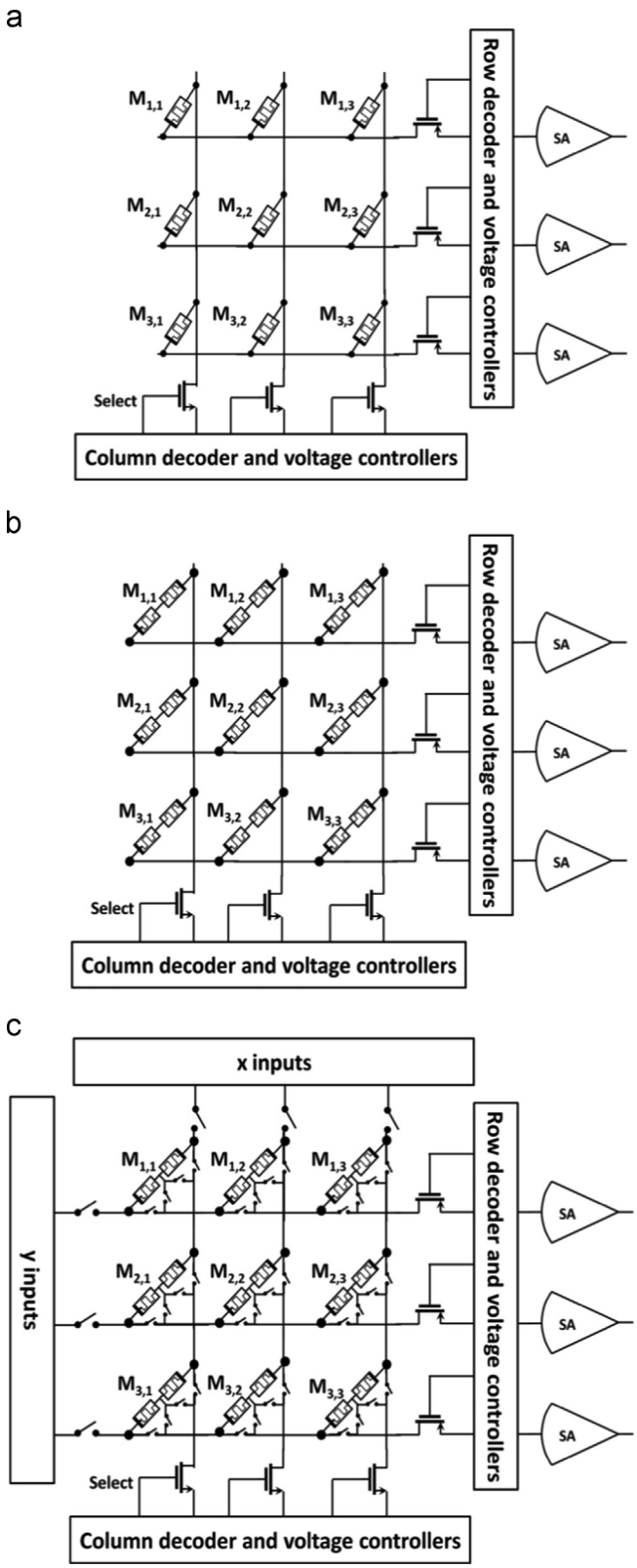

Fig. 7. Memristive memories (exemplified by a three by three array). (a) Single memristor within a crossbar, (b) standard complementary memristive cells within a crossbar, and (c) Akers logic array within a memristive memory. The basic memory cell for the Akers logic array consists of two memristors and four CMOS transistors (as selectors).

2) Stage 2 – execution of the Boolean function (read)
The structure of the memristive logic array is shown in Fig. 2a. The array is similar to the structure of the original Akers logic array. In a memristive Akers logic array, each primitive logic cell consists of complementary memristors. The x and y control inputs are voltages, and, as in the original Akers array, the input y of the left-most column is set to logical one (execution voltage V_r), and the input x of the upper row is set to logical zero (ground) for all columns. Since the output of the memristive primitive logic cell is a voltage, the result of the logical operation for each primitive logic cell is transferred to the neighboring cells.

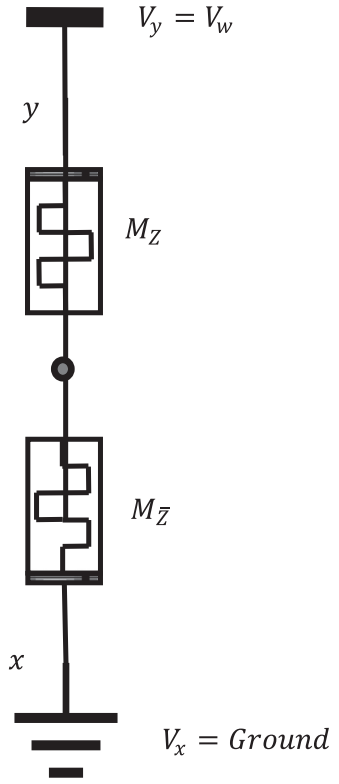

Fig. 8. Write operation of logical one to memristor M_Z. Due to the complementary structure of the circuit, writing to M_Z and $M_{\bar{Z}}$ is achieved simultaneously in both memristors by applying a single voltage V_W. After the write procedure, the resistance of M_Z and $M_{\bar{Z}}$ is, respectively, R_{ON} and R_{OFF}.

To maintain correct operation of the memristive Akers logic array, the resistance of the memristors in the array must not change during execution. The current flowing through the memristors I_r is therefore maintained lower than the threshold current of the memristors. The current is

$$I_r = \frac{|V_y - V_x|}{R_Z + R_{\bar{Z}}} \leq \frac{V_r}{R_{ON} + R_{OFF}} < \max(|i_{off}|, |i_{on}|). \quad (5)$$

4. Evaluation of primitive logic cells

In this section, the proposed memristive primitive logic cell is evaluated with 0.18 μm CMOS and simulated in SPICE. A Verilog-A TEAM model [17] is used to simulate the behavior of the memristors.

The primitive logic cell is based on a complementary resistive switch structure. The CRS behaves as a linear resistor with a resistance of $R_{ON}+R_{OFF}$ below a certain voltage. Above this voltage, hysteresis exists in the current–voltage curve of the CRS [15,16]. The current–voltage curve of the primitive logic cell is shown in Fig. 9.

The primitive logic cell is evaluated with and without CMOS selectors connected to the control inputs, x and y. The primitive

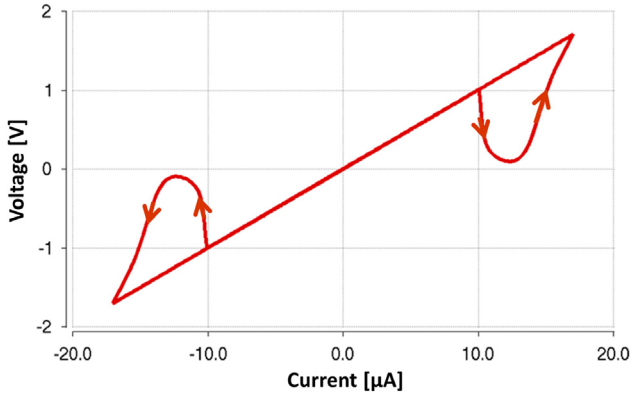

Fig. 9. Current–voltage characteristic of the primitive logic cell for a sinusoidal current input with an amplitude of 17 μA and frequency of 100 kHz. The circuit parameters are listed in Table 2. For a current lower than the current thresholds i_{on} and i_{off} (10 μA), the resistance of both memristors is constant. For a current higher than the current thresholds, the resistance of both memristors changes.

Table 3
Memristor parameters.

k_{on}	−8 m/s
k_{off}	0.5 m/s
i_{on}	−10 μA
i_{off}	10 μA
x_{on}	0
x_{off}	3 nM
α_{on}	1
α_{off}	4
R_{ON}	100 Ω
R_{OFF}	100 kΩ
V_w	3 V
V_r	1 V
CMOS selectors	CMOS 0.18 μm process W=0.42 μm

logic cell drives a load capacitor of 10 fF. The parameters used for the memristors are listed in Table 3. A schematic of the simulated primitive logic cell is shown in Fig. 10a. The results of the initializing stage are shown in Fig. 10b. The write latency of the primitive cell depends upon the switching time of the memristor, assumed as 1.1 ns. The primitive logic cell exhibits a write latency of 6.6 ns (six times more than the switching time of a single memristor).

The results of the execution stage are shown in Fig. 10c and 10d. The primitive logic cell executes the correct logical behavior with degradation in the output signal. The degradation depends upon the ratio between R_{OFF} and R_{ON}. The output degradation is 0.1% without selectors (R_{OFF}/R_{ON}=1000) and 4% with CMOS selectors (for a 0.18 μm CMOS process). The output degradation is discussed in the following section.

5. Output degradation

Since memristors are passive elements, signal degradation occurs at the output of each primitive logic cell. The degradation depends primarily on the ratio between R_{OFF} and R_{ON}, where a higher ratio reduces the degradation. The degradation limits the size of the Akers array.

The degradation of the output signal as a function of array size is shown in Fig. 11a for Akers arrays with and without CMOS selectors. The use of CMOS selectors makes the output degradation worse since the CMOS element adds a resistance in series. For larger arrays, the degradation is more significant and limits the size of the sub-arrays of the memory. The degradation for different

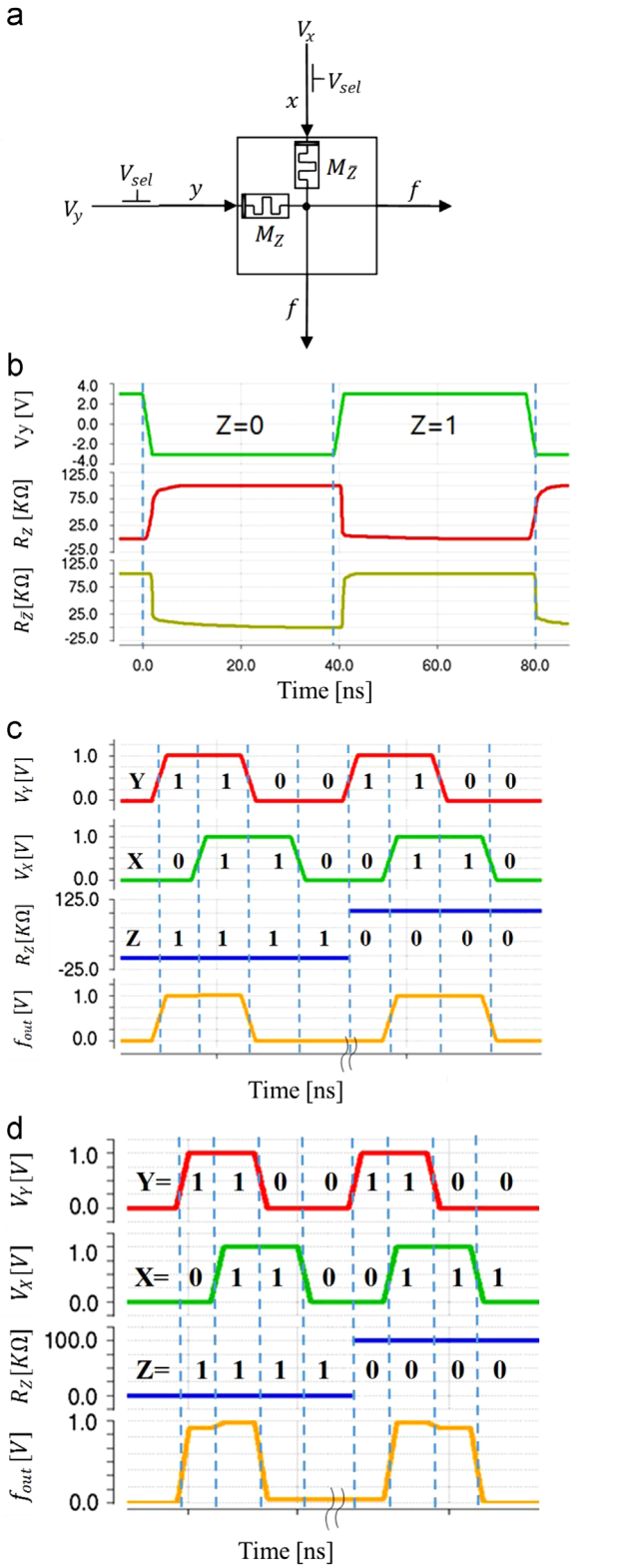

Fig. 10. Initialization and execution of primitive logic cell. (a) Schematic of the simulated circuit, (b) simulation of memristive initialization operation. V_y is the write voltage applied to the primitive logic cell (positive and negative for, respectively, writing logical one and zero to Z), and simulation of memristor execution operation (c) without selectors and (d) with selectors. The simulation parameters are listed in Table 2.

ratios of R_{OFF} and R_{ON} is shown in Fig. 11b. For an array composed of 128 by 128 primitive logic cells, the minimal degradation of the output reaches 10% for R_{OFF}/R_{ON}=1000. For arrays with CMOS

selector with a resistance of 1 kΩ, the actual output degradation is 15%. Using larger CMOS transistors lowers the degradation. A higher R_{OFF}/R_{ON} ratio enables a larger array, where a ratio of 10,000 enables arrays of more than a million logic primitive cells with an output degradation of 10%.

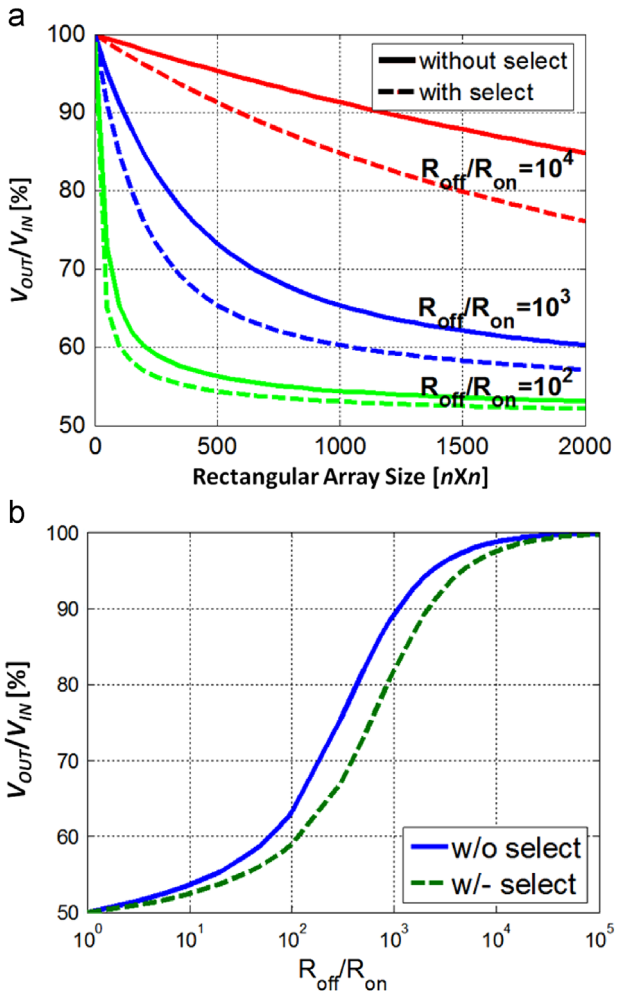

Fig. 11. Output signal degradation for an Akers array with (dashed line) and without (solid line) CMOS selectors. (a) Signal degradation as a function of rectangular array size for different R_{OFF}/R_{ON} ratios (10^4 in red, 10^3 in blue, and 10^2 in green), and (b) signal degradation in rectangular array of 128 by 128 as a function of the resistance ratio R_{OFF}/R_{ON} with CMOS selector. $R_{ON}=1$ kΩ, the resistance of a CMOS selector is 1 kΩ. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6. Test case – memristor-based logic within memory array

To evaluate a memristive Akers array, several Boolean functions are investigated within the array. In this section, simulation results of a two-input XOR and sorting of four bits are presented as simple test examples.

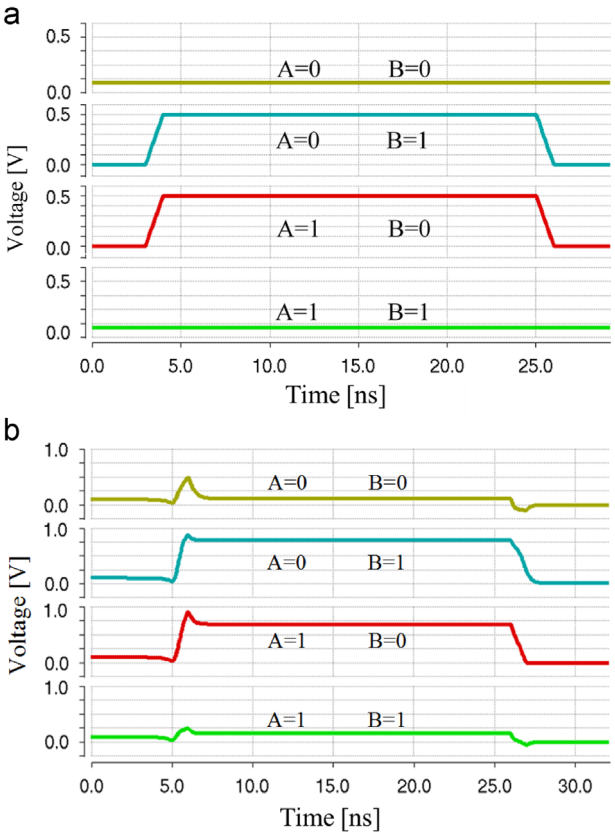

Fig. 13. Simulation results of a two-input XOR (a) without CMOS selectors and (b) with CMOS selectors for different inputs *A* and *B*. The average output degradation is 3% and 20%, respectively, without and with CMOS selectors for a 0.18 μm CMOS process. The execution voltage V_r for the XOR without selectors is 0.5 V.

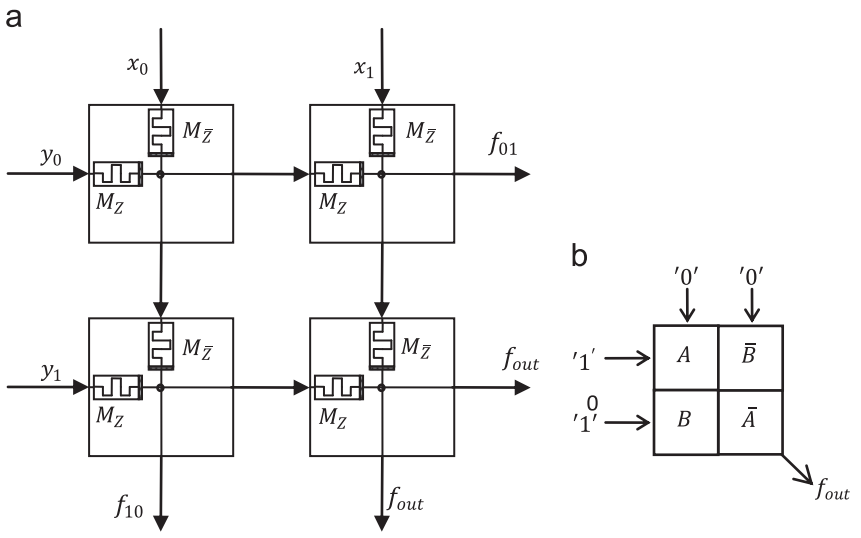

Fig. 12. Two-input XOR. (a) Schematic of a two by two memristive Akers array, and (b) the array structure of the Boolean function XOR(*A*, *B*).

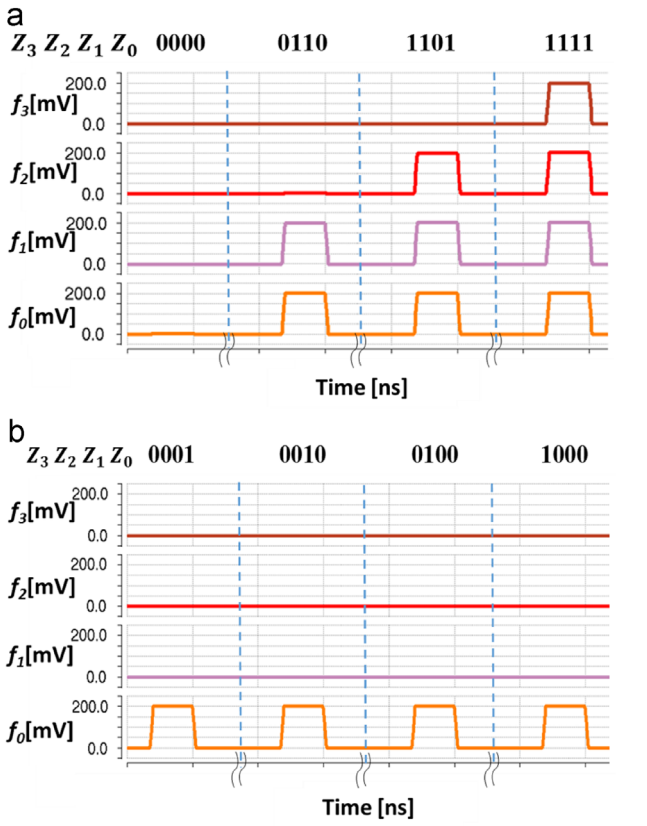

Fig. 14. Simulation results of a four-bit set sort using a four by four memristive Akers array without CMOS selectors. (a) Different output values and (b) different inputs, all with a single logical one and three zeros. The output is therefore the same for all input cases. The execution voltage V_r is 200 mV.

6.1. Two-input XOR

The schematic and array structure of a $XOR(A, B)$ are shown in Fig. 12. The memristive Akers array is a two by two array, consisting of eight memristors. Initializing the array (writing the inputs to the memristors) is achieved prior to execution. The execution is evaluated with the same parameters listed in Table 3, exhibiting the correct output. The average and maximum output degradation are, respectively, 20% and 31% for a two-input XOR with 0.18 μm CMOS selectors (3% without selectors). The relatively high degradation is due to the minimal size of the CMOS selectors and the use of high voltage transistors, which have a relatively high resistance. As previously mentioned, increasing the width of the transistors significantly lowers the signal degradation.

The average power of the array during execution is, respectively, 6.2 μW and 33.6 μW without and with CMOS selectors. The results for different input conditions are shown in Fig. 13. For small arrays, adding CMOS selectors does not affect the speed of the circuit. For an array with CMOS selectors, execution is slower due to the capacitance of the selectors.

6.2. Sorting of bits

To evaluate sorting of bits, a four-bit sorting Boolean function is executed within the memristive Akers array. The memristive Akers array consists of 10 primitive logic cells (see Fig. 3a) and 20 memristors. The execution is evaluated with the same parameters listed in Table 3, showing correct output and an average output degradation of 0.3% without CMOS selectors. The average power of the array during execution is 1.6 μW. Results for different input conditions are shown in Fig. 14.

7. Conclusions

The proposed memristive Akers array contains a pair of complementary memristors in each cell. The array can therefore be used as a memristive memory, where a single bit is stored within a memristor pair rather than a single memristor [15,16]. Each cell also performs a primitive Boolean operation, which enables the logic functionality of the array, as initially shown by Akers. The combination of an Akers array and memory is promising and may lead to additional uses, as described in [18]. For example, an Akers logic array naturally performs bit sorting which may lead to efficient sorting of words and other data structures.

The integration of memristive memory with a logic array that executes any Boolean function can lead to a variety of novel non-von Neumann architectures. The Akers array architecture eliminates the memory bottleneck, reducing power and bandwidth. Memristive Akers logic arrays may also be beneficial for image processing applications and error correcting operations within memory.

Acknowledgments

The authors thank Ravi Patel of the University of Rochester for his useful comments.This work was partially supported by Hasso Plattner Institute, by the Advanced Circuit Research Center at the Technion, by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and by US-Israel Binational Science Foundation under Grant no. 2012139.

References

[1] S.B. Akers Jr., A rectangular logic array, IEEE Trans. Comput. C-21 (8) (1972) 848–857 (August).
[2] H.T. Kung, Why systolic architectures? IEEE Comput. 15 (1) (1982) 37–46 (January).
[3] L.O. Chua, Memristor – the missing circuit element, IEEE Trans. Circuit Theory 18 (5) (1971) 507–519 (September).
[4] S. Kvatinsky, N. Wald, G. Satat, E.G. Friedman, A. Kolodny, and U.C. Weiser, MRL – memristor ratioed logic, in: Proceedings of the International Cellular Nanoscale Networks and their Applications, August 2012, pp. 1–6.
[5] S. Kvatinsky, N. Wald, G. Satat, E.G. Friedman, A. Kolodny, U.C. Weiser, Memristor-based material implication (IMPLY) logic: design principles and methodologies, IEEE Trans. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, (2014), vol. PP, no.99, pp.1,1, 0.
[6] S. Kvatinsky, E.G. Friedman, A. Kolodny, and U.C. Weiser, Memristor-based IMPLY logic design procedure, in: Proceedings of the IEEE International Conference on Computer Design, October 2011, pp.142–147.
[7] L.O. Chua, S.M. Kang, Memristive devices and systems, Proc. IEEE 64 (2) (1976) 209–223 (February).
[8] D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found, Nature 453 (2008) 80–83 (May).
[9] X. Wang, Y. Chen, H. Xi, D. Dimitrov, Spintronic memristor through spin-torque-induced magnetization motion, IEEE Electron Dev. Lett. 30 (3) (2009) 294–297 (March).
[10] L.O. Chua, Resistance Switching Memories are Memristors, Appl. Phy. A: Mater. Sci. Process. 102 (4) (2011) 765–783 (March).
[11] R. Waser, R. Dittmann, G. Staikov, K. Szot, Redox-based resistive switching memories – nanoionic mechanisms, prospects, and challenges, Adv. Mater. 21 (25–26) (2009) 2632–2663 (July).
[12] J.J. Yang, D.B. Strukov, D.R. Stewart, Memristive devices for computing, Nature Nanotechnol. 8 (2013) 13–24 (January).
[13] S. Kvatinsky, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, MRL – memristor ratioed logic, Proceedings of the International Cellular Nanoscale, Networks and their Applications, (2012), pp. 1-6.
[14] S. Kvatinsky, E.G. Friedman, A. Kolodny, U.C. Weiser, TEAM – ThrEshold adaptive memristor model, IEEE Trans. Circuits Syst. I: Regul. Pap. 60 (1) (2013) 211–221 (January).
[15] E. Linn, R. Rosezin, C. Kügeler, R. Waser, Complementary resistive switches for passive nanocrossbar memories, Nat. Mater. 9 (5) (2010) 403–406 (April).
[16] O. Kavehei, S. Al-Sarawi, S., K.-R. Cho, K. Eshraghian, D. Abbott, An analytical approach for memristive nanoarchitectures, IEEE Trans. Nanotechnol. 11 (2) (2012) 374–385 (March).

[17] S. Kvatinsky, K. Talisveyberg, D. Fliter, E.G. Friedman, A. Kolodny, and U.C. Weiser, Models of memristors for SPICE simulations, in: Proceedings of the IEEE Convention of Electrical and Electronics Engineers in Israel, November 2012,.pp.1–5.
[18] E. Yaakobi, A. Jiang, and J. Bruck, In-memory computing of Akers logic array, in: Proceedings of the IEEE International Symposium on Information Theory, July 2013, pp. 2369–2373.
[19] S. Wolfram, Universality and complexity in cellular automata, Phys. D: Nonlinear Phenom. 10 (1–2) (1984) 1–35 (January).
[20] E. Gale, B.d.e. Lacy Costello, and A. Adamatzky, Boolean logic gates from a single memristor via low-level sequential logic, in: Proceedings of the International Conference on Unconventional Computation and Natural Computation, July 2013, pp. 78–89.
[21] E. Gale, B.d.e. Lacy Costello, and A. Adamatzky, Is spiking logic the route to memristor-based computers? in: Proceedings of the International Conference on Electronics, Circuits and Systems, December 2013, pp. 297–300.
[22] M.D. Pickett, R.S. Williams, Phase transitions enable computational universality in neuristor-based cellular automata, Nanotechnology 24 (38) (2013) 1–7 (September).
[23] S. Shin, K. Kim, S.-M. Kang, Memristive XOR for resistive multiplier, Electron. Lett. 48 (2) (2012) 78–80 (January).
[24] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, R. Waser, Beyond von Neumann – logic operations in passive crossbar arrays alongside memory operations, Nanotechnology 23 (305–205) (2012) (August).
[25] J. Borghetti, G.S. Snider, P.J. Kuekes, J.J. Yang, D.R. Stewart, R.S. Williams, Memristive switches enable 'stateful' logic operations via material implication, Nature 464 (2010) 873–876 (April).
[26] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E.G. Friedman, A. Kolodny, U.C. Weiser, MAGIC – memristor aided LoGIC, IEEE Trans. Circuits Syst. II: Express Briefs (2014) (to appear).