Unconventional digital computing approach: memristive nanodevice platform

Mahyar Shahsavari, M Faisal Nadeem, S Arash Ostadzadeh, Philippe Devienne, Pierre Boulet

Unconventional digital computing approach: memristive nanodevice platform

Mahyar Shahsavari[*,1], M. Faisal Nadeem [2], S. Arash Ostadzadeh[2], Philippe Devienne[1], Pierre Boulet[1]

[1] University of Lille, LIFL, CNRS, UMR 8022, F-59650 Villeneuve d'Ascq, France
[2] Department of Software & Computer Technology, Delft University of Technology, Netherland

Key words: material implication, nanoscale crossbar, memristor nanodevice, digital computing

[*] Corresponding author: e-mail mahyar.shahsavari@lifl.fr, Phone: +33 (0)3 20 33 59 51

Memristor is a two-terminal nanodevice that has recently attracted the attention of many researchers. Its simple structure, non-volatility behavior, high-density integration, and low-power consumption make the memristor a promising candidate to act as a switch in digital gates for future high-performance and low-power nanocomputing applications. In this paper, we model the behavior of memristor by using Verilog-A. To investigate its characteristics in a circuit, we use the HSPICE simulator. Furthermore, a library of digital gates are provided by using two approaches to make digital gates: the first one is based on material implication (IMP) and the second one is based on crossbar arrays. Finally, we perform a comparison and evaluation between the two methods.

1 Introduction Memristor has recently drawn wide attention of scientists and researchers due to non-volatility, better alignment, and excellent scalability properties [1]. Memristor remembers its last state after the last power plugging and has simple physical structure, high-density integration, and low-power consumption. These features make memristor an attractive candidate for building the next generation non-volatile memories [2]. From high-performance computing point of view, memristor has potential capability to conquer the memory bottleneck issue, by utilizing computational unit next to the memory [3].

The logic computing applications of memristor have been investigated by several researchers [4–13]. For instance, Borghetti et al. [12] used *material implication* (IMP) logic operation to carry out logic computation by using memristors. In IMP approach, A IMP B ($A \rightarrow B$) operation means 'if A, then B' and can be read as A implies B. IMP together with a FALSE gate are able to form a functionally complete set (any boolean function can be expressed). By applying memristor as a digital switch, a high memristance (memristor resistance) is considered as logic '0' and a low memristance is considered as logic '1'. Another approach to make gates by a two-terminal device as a switch is the *programmable crossbar architecture* [8,13]. The crossbar nanowire array architecture can be used to compute logic functions by using memristor as a switch between two nanowires [4].

The main contributions of this paper are:

– To demonstrate that memristor has appropriate characteristics to be applied as a switch in both IMP and crossbar array methods, we proposed our electrical model of the memristor behavior. Modeling is performed by using Verilog-A Hardware Description Language (HDL).
– Material implication (IMP) logic function and crossbar array architecture are investigated as the two novel and most promising methods for creating logical functions by memristors.
– The crossbar array architecture approach is developed in the term of using only memristors as switches between nanowires.

The reminder of the paper is organized as follows. The electrical model is presented in Section 2. Then, we apply

our Verilog-A model in the HSPICE circuit simulator to observe memristor behavior in the circuits. In Section 3, the stateful logic operations via IMP is described. In Section 4, the crossbar array architecture is presented by using memristor as a junction switch. Evaluations of the two approaches are investigated in Section 5. Section 6 concludes the paper.

2 Electrical model When an electric field is applied to the terminals of memristor, the shifting in boundary between its doped and undoped regions leads to variable total resistance of the device. In Figure 1.a, the electrical behavior of memristor can be modeled as follows [14]:

$$v(t) = R_{mem} i(t) \tag{1}$$

$$R_{mem} = R_{ON}\frac{w(t)}{D} + R_{OFF}(1 - \frac{w(t)}{D}) \tag{2}$$

where $w(t)$ is the width of the doped region, D is the overall thickness of the TiO_2 bi-layer, R_{ON} is the resistance when the active region is completely doped ($w = D$) and R_{OFF} is the resistance, when the TiO_2 bi-layer is mostly undopped ($w \rightarrow 0$).

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

which yields the following formula for $w(t)$:

$$w(t) = \mu_v \frac{R_{ON}}{D} q(t) \tag{4}$$

Where μ_v is the average dopant mobility. By inserting Equation (4) into Equation (2) and then into Equation (1) we obtain the memristance of device, which for $R_{ON} \ll R_{OFF}$ simplifies to:

$$M(q) = R_{OFF}(1 - \frac{\mu_v R_{ON}}{D^2} q(t)) \tag{5}$$

Equation (5) shows the dopant drift mobility μ_v and semiconductor film thicknesses D values, which are two factors with crucial contributions to the memristance magnitude. Subsequently, we can write Kirchoff's voltage law for memristor given by:

$$v(t) = M(q) i(t) \tag{6}$$

By using Verilog-A HDL, we simulate the behavior of memristor, based on its behavioral equations. To investigate the characteristics of memristor in electrical circuits, the Verilog-A model of memristor behavior must be applied as a circuit element in the HSPICE netlist. In the HSPICE circuit, we apply a sinusoidal source to observe the memristor reaction in a simple circuit consisting of memristor and sinusoidal source. Figure 1.b depicts $i - v$ plot of memristor terminals that we measured in our simulation. This $i - v$ plot, which is the most significant feature of memristor [15], is namely called "pinched hysteresis loop". The $i - v$ characteristic demonstrates that memristor can [U+201C]remember[U+201D] the last electric charge flowing across it by changing its memristance. Therefore, we can use memristor as a latch to save data and also as a switch for computing. Moreover, in Figure 1.b, it is depicted that the pinched hysteresis loop is shrunk by increasing frequency. In fact, when the frequency increases toward infinity, memristor behaves similar to a linear resistor.

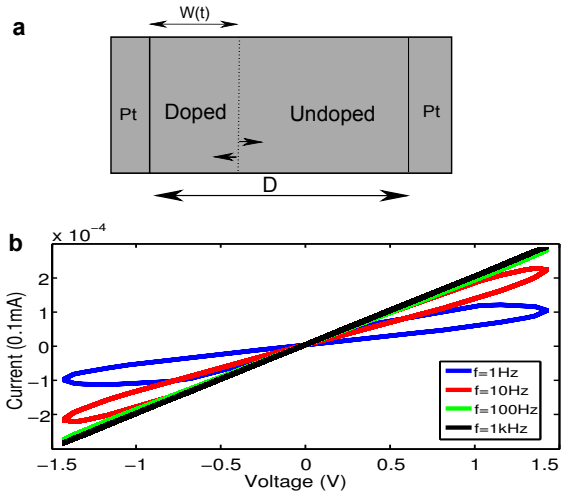

Figure 1 Memristor schematic and behavior: **a)** the memristor structure, the difference in applied voltage changes doped and undoped regions, **b)** current versus voltage diagram, which demonstrates hysteresis characteristic of memristor, in the simulation we apply the sinusoidal input wave with an amplitude of 1.5v, different frequencies, $R_{ON} = 100\Omega, R_{OFF} = 15k\Omega, D = 10nm, \mu_v = 10^{-10} cm^2 s^{-1} V^{-1}$.

3 Stateful implication logic One of the basic potential applications of memristors is to utilize them in building blocks of logic gates. Therefore, by applying a digital pulse voltage to the memristor terminal, we have a switch with ON or OFF state. Unlike conventional CMOS, in memristor-based gates, data will be stored as a resistance rather than a voltage. In this case, the latches are non-volatile. Thus, R_{ON} displays logic '1' which means closed switch and R_{OFF} displays '0' for presenting open switch. In contrast to the three-terminal CMOS-based transistor as a switch, in a two-terminal switch, there is no terminal to control ON or OFF states of the switch. Consequently, instead of conventional boolean logic, we should find other substitutes to create a gate and perform computing.

IMP (Figure 2.a) is a way to use one memristor to control the other one. IMP is recognized as a promising method for making gates by memristors [9–12]. In IMP structure, memristors have different roles in different

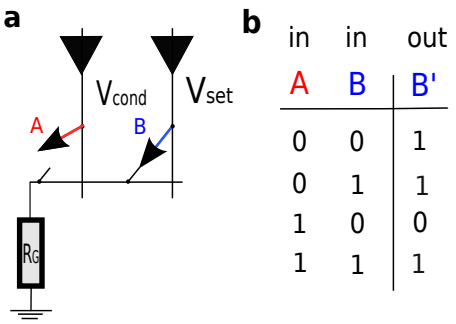

Figure 2 Memristor-based IMP: **a)** circuit schematic, **b)** IMP truth table.

stages of the computing process: input, output, computational logic element, and a latch depending upon which write, read, computing and storing processes are taking place, respectively.

To figure out how IMP operates, imagine A as a question and B as the answer to that question. If the question is wrong, any answer (wrong or correct) makes a true output (logic 1), as depicted in Figure 2.b. The only case for the false (logic 0) output would be a wrong answer to a correct question. So the implication logic is equivalent to function $(\neg A) \vee B$. Figure 2.a shows the basic circuit of memristors A and B to perform implication logic, which are formed by the vertical nanowire crossing over the horizontal nanowire connected to a load resistor R_G. After the operation of material implications, the result is stored as the state of switch B (B') while A is left unchanged. To switch between logic 1 to logic 0 (and vice versa), we need a tri-state voltage driver with a high impedance output state when it is undriven. V_{set} is a negative voltage which should be applied to its corresponding tri-state driver. V_{set} can switch memristor to conductive state with low resistance R_{ON}. Similarly, the positive voltage V_{clear} is required to change the memristive switch state to low-conductance (high-resistance) state R_{OFF}. It is important to mention that the magnitude of V_{set} and V_{clear} must be larger than device threshold voltage for switching ON and OFF. In order to remain in a specific state (line 3 in truth table 2.b), the V_{cond} is applied as a negative voltage with a magnitude smaller than V_{set}. Consequently, tri-state drive —since is not in high-impedance state— is pulsed by one of the V_{set}, V_{clear} or V_{cond}. By applying V_{cond} and V_{set} to A and B simultaneously, the memristive IMP operates properly. Although the conditional voltage (V_{cond}) is not necessary, except for the case AB='10' (third line in the truth table Figure 2.b), it is possible to either apply V_{cond} or use high-impedance (HZ) for all other cases. If V_{set} is applied to B alone, it would be unconditionally 'logic 1', nevertheless applying V_{cond} by itself to A does not change its state. On the other hand, if both voltages are applied together, the present state of switch A determines the next state of switch B. If A='0', it means memristor A is in high resistance state (R_{OFF}). Therefore, there is a small voltage drop across R_G. In this case, B will be set and A is left unchanged. Alternatively, if present state of A='1', switch A is in low resistance state and V_{cond} drops across R_G, so both A and B remain unchanged. It should be noted that the R_G value must be chosen such that $R_{ON} < R_G < R_{OFF}$, where R_{ON} and R_{OFF} are resistance states of 'ON' and 'OFF' switches, respectively.

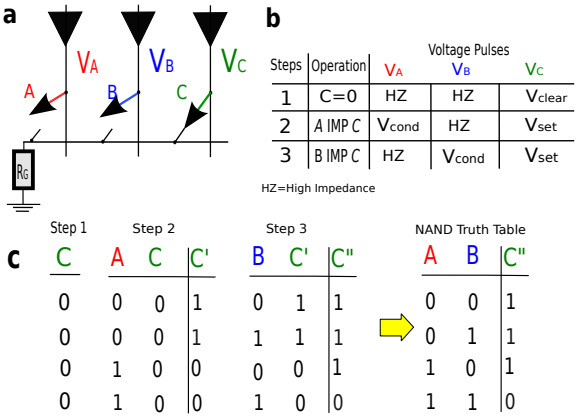

Figure 3 NAND configuration with IMP: **a)** circuit schematic, **b)** required voltages for controlling the process, **c)** sequential truth table to obtain NAND.

In boolean logic algebra, a set of logic operations is called functionally complete if all other logic functions can be constructed from combining the members of this set. The single-element sets *NAND* (AND, NOT) and *NOR* (OR, NOT) are functionally complete operations. In addition, it has been demonstrated that all boolean expressions can then be written in one of the standard normal forms using only a (IMP) operation and a *false* (Inverse) operation [11,12]. In fact, to show that the memristive implication logic is a functionally complete operation; the easiest way is to synthesize the NAND function with it.

A circuit with three memristors A, B, and C is illustrated schematically in Figure 3.a. Assume two implication operations being performed subsequently, first A IMP C and then B IMP C. Hence A and B are inputs and C is output. The final output result is represented with variable C$''$ in Figure 3.c. Firstly, V_{clear} should be applied to switch C to create the *false* operation ($C' = (C \text{ IMP } 0)$). By applying V_{cond} and V_{set} pulses to A and C respectively, the second step would be performed. Finally, $C'' = (B \text{ IMP } C)$ is yielded by applying V_{cond} to V_B and V_{set} to V_C (see Figure 3.b and Figure 3.c). In other words, the resulting state of switch C can be written as:

$$C = B \text{ IMP } (A \text{ IMP } C) = \neg B \vee ((\neg A) \vee C)$$

if $C = 0$ initially then

$$C = \neg B \vee ((\neg A) \vee 0) = (\neg B) \vee (\neg A)$$
$$= \neg(A \wedge B)$$

which is a NAND operation. Similarly, we can produce equal IMP structure of other logic operations. In Table 1 different boolean logic gates are listed. Obviously, we can demonstrate all logic relations in Table 1, not only by applying boolean logic rules but also by checking the truth tables of both sides of relations.

Table 1 Different logic operations made by IMP operations and the number of required memristors

Logic Operation	Equal IMP functions	#Device
NOT A	A IMP 0	3
A AND B	$\{A$ IMP (B IMP 0)$\}$ IMP 0	4
A NAND B	A IMP (B IMP 0)	3
A OR B	(A IMP 0) IMP B	3
A NOR B	$\{(A$ IMP 0) IMP $B\}$ IMP 0	4
A XOR B	(A IMP B) IMP $\{(B$ IMP $A)$ IMP 0$\}$	3

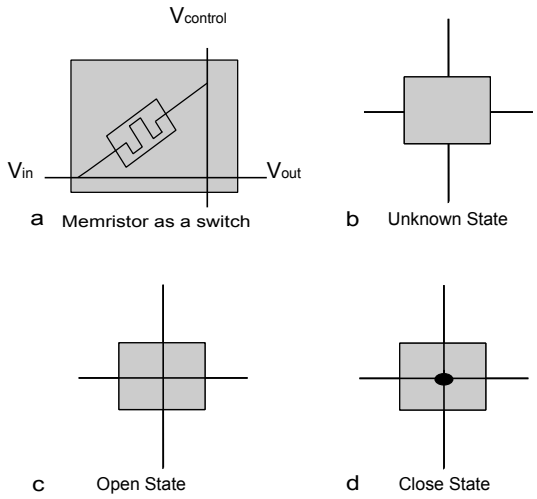

Figure 4 Different states of a switch in a crossbar array.

4 Crossbar architecture Programmable crossbar architectures have been proposed as a promising approach for future computing architectures because of their simplicity of fabrication and high density, which support defect tolerance [13,16]. In such architectures, assume that each junction within the crossbar can be utterly configured to activate an electronic device, such as a resistor, diode, transistor, or recently memristor. In fact, attractive features of memristor, such as simple physical structure, non-volatility, high-density, and unlimited endurance make this nano-device one of the best choices to play a switch role in the crossbar junctions. The memristive-based crossbar opens new windows to explore advanced computer architecture, different from the classical Von Neumann architecture [16]. On the other hand, in crossbar architecture, memory and logic operators are not separated. The memory can perform logic implementations on the same devices which store data. This is because during the operation process, control signals determine which elements act as logic gates and which ones act as memory cells.

4.1 Memristive switches in crossbar architectures Each junction in a crossbar could be connected or disconnected by replacing a memristor as a switch in the junction point between two vertical and horizontal nanowires as depicted in Figure 4.a. Such a switch is in high resistance (R_{OFF}) (Figure 4.c) for open state or low resistance (R_{ON}) (Figure 4.d) for close state, similar to the states of memristor in Section 3. The memristive switch retains its state as long as the voltage drop across it, is not more than the required threshold voltage to change the memristor state. In Figure 4.a the input voltage (v_{in}) either could be connected to the output voltage of an external CMOS circuitry or be connected to the output of another latch from the previous stage. As we have already mentioned, data in our architecture is saved as the resistance of the memristive latch. However, for data transferring, an input voltage would be necessary. This input voltage is driven by the state of the input impedance (R_{in} in Figure 5), which is a memristive switch. This voltage can be disconnected which means the floating state (for instance, a very high impedance $\gg R_{OFF}$). The floating state input happens when the input memristor (R_{in} in Figure 5.a) is OFF. Subsequently, the input voltage can also be a fixed negative voltage ($-V_f$) with less magnitude than the threshold, while the input memristor (R_{in} in Figure 5.b) is ON.

The memristive switches can be configured as an inverting or non-inverting mode as it is depicted in Figure 5.c and Figure 5.d, respectively. If the stored data in the receiving switch is the logical complement of the input data, the configuration is in the inverting mode. If the stored data is the same as the input data the configuration is in the non-inverting mode. Different control signals make different configurations. Inverting configuration requires three steps to perform the appropriate latch operation:
1) We preset the switch unconditionally open (Figure 5.a) by applying a positive voltage (more than the positive threshold) to the control of the vertical nanowire and also by forcing input to the high impedance mode. A diode is also required to provide a low-impedance path to the ground to protect the junction.
2) If the input voltage is logic '1' then R_{in}=R_{ON} and $-V_f$ drops across R_G, so the voltage across memristor is not enough to close it and the switch remains OFF. On the other hand, when input is logic '0', then R_{in}=R_{OFF} and memristor switch turns ON. Thus, the junction has held the inverted state of the input (see Figure 5.b).
3) The input signal must be in high impedance (disconnected). The state of the switch is read out onto the horizontal nanowire. This is accomplished by driving the vertical nanowire with a small voltage whose magnitude is less than the threshold control voltage (can not change the

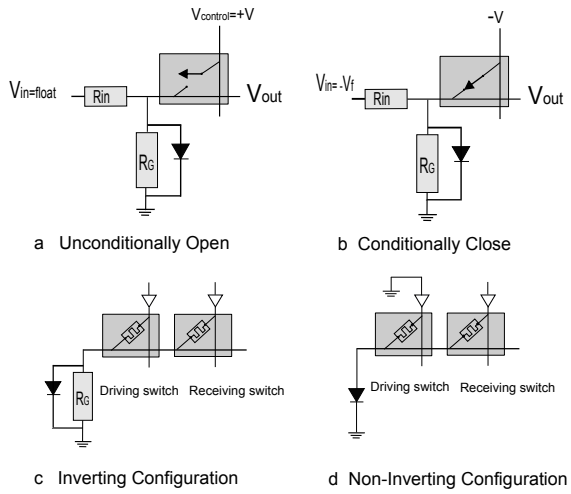

Figure 5 Different states and configurations of the memristive switches in a crossbar array.

switch state). We call this voltage v_R (Read voltage).
For non-inverting configuration, as depicted in Figure 5.d, the pull-down resistor R_G is not required when writing the state of the driving switch to the receiving one. The steps for appropriate latch operation are the same as the 'inverting configuration' except grounding the vertical nanowire of the driving switch rather applying a negative voltage in conditionally close step. In this case, by applying negative voltage to the vertical control line, the receiving switch will be close if the driving switch is close (low-impedance path to the ground) and the receiving switch remains open if the driving switch is open. Thus, the state of the first switch is replicated in the second switch.

4.2 Configurable crossbar crray logic gates In this section, we demonstrate how to make a gate by using the crossbar approach. We start using crossbar to make a 3-input AND gates. Figure 6.a shows a crossbar array which consisting of 8 memristors. We need three inputs A, B, and C, each of which is assumed to be impedance encoded. Basically, various voltages are applied to vertical nanowires to control the memristive switches. The inputs are connected to the horizontal wires as represented in Figure 6.a. The crossbar array functions as an AND gate by applying the following sequences.

1. All junctions are unconditionally opened by applying a positive voltage higher than the threshold voltage to the v_{in}, v_{and}, v_{out} control vertical lines.
2. By driving v_{in} with a negative voltage, the input data (A, B, C) are latched in the input switches. The inputs are in the inverting configuration thus '0' and '1' inputs, close and open the switches, respectively (Figure 6.b).
3. In this step, by driving the v_{in} and K input to the ground, as well as v_{and} to the negative voltage (Figure 6.c) the input data are latched to the wired-AND junctions.
4. In this step, the vertical control lines of the input and output (v_{in}, v_{out}) are activated by a negative voltage to capture the result to the output memristor (S8). The voltage in point X in Figure 6.d can determine output switch state. If there is at least one closed input (also wired-AND) in the route from v_{in} to the X point, the negative voltage efficacy causes the output switch to stay open. The reason is that the potential difference across the output switch is not enough to close it. It is noteworthy to mention that the voltage at X point is yielded from the voltages dividing between the resistance of switches in the route and S7 that connected to the ground in the last horizontal nanowire line.
5. By driving a positive voltage to v_{and}, all junctions in this column will be opened (Figure 6.e).
6. The AND of the three inputs is stored in the output switch and is ready to be read out.

If the output switch is at inverse mode (inverting configuration), the crossbar array becomes a 3-input NAND gate. Moreover, the crossbar array can also become a 3-input NOR gate, if R_G's are removed while input data are applied to the circuit (Non-inverting configuration). By inverting the last output with the recent situation of the inputs, the crossbar architecture becomes a 3-input OR gate. Consequently, all logic gates are created except XOR and XNOR, which require a little different structures. First, we create an exclusive-NOR gate, subsequently, by inverting the output, XOR is obtained. To implement XNOR, two minterms should be OR'ed together. The crossbar array operation for creating XNOR is similar to AND crossbar array in Figure 6. However, three additional steps are required. To simplify the crossbar array, we apply two variables A and B (Figure 7). In the following, we discuss these sequences step by step.

1. All junctions are unconditionally opened by applying a positive voltage to the v_{in}, two v_{AND}'s, and v_{out} control lines.
2. By driving v_{in} with a negative voltage, the input data (A, B, $\bar{A}$, $\bar{B}$) are latched in the input switches. The inputs are in the inverting configuration, therefore, '0' and '1' inputs close and open the switches, respectively.
3. In this step, by driving the v_{in} and K input to the ground, and v_{AND1} to the negative voltage, the input data are latched to the wired-AND for the first minterm.
4. In this step, the vertical control lines of input and output (v_{in}, v_{out}) are activated by a negative voltage for first minterm to capture the result ($\bar{A}\bar{B}$) to the output.
5. By applying the positive voltage to v_{AND1} (the wired-AND), junctions become open for the first minterm.
6. For the second minterm implementation, step 3 should be repeated by applying a negative voltage to v_{AND2}.
7. In this step, the control lines of input and output (v_{in}, v_{out}) are activated by a negative voltage for the second

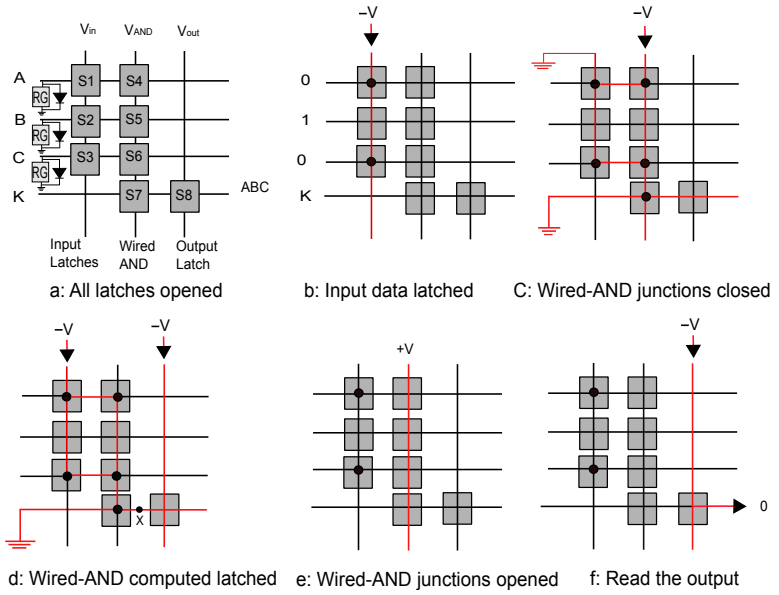

Figure 6 The crossbar array architecture for AND function with memristor switches: **a)** by applying the positive voltages all switches become open (clear), **b)** capture data in the input latches, **c)** transfer data to the wired-AND switches, **d)** if all inputs are '1' (open) then the X spot is not negative so the output switch has enough voltage across it to be ON, **e)** open all wired-AND switches to be prepared to read output, **f)** read output.

minterm to capture the result (AB) to the output. It is worth to note that performing OR of the two minterms is sequential rather than concurrent. It means that if the output switch is set into the close state by the first minterm, the value of the second minterm has no effect. Otherwise, changing the state of the output switch is dependent on the second minterm.

8. By applying the positive voltage to the v_{AND2}, the wired-AND junctions become open for the second minterm.
9. The XNOR of two inputs A and B is stored in the output switch and it is ready to be read out.

It is also possible to make other boolean logic functions by applying NAND combinations. However, it requires a larger number of memristive switches. Furthermore, it nullifies the most important capability of memristors, i.e. configurability.

5 Evaluation In this section, we present an evaluation of both the IMP and crossbar array approaches. Both approaches have significant advantages over the CMOS-based logic gates, such as having computing and storing on the same physical unit, non-volatility, and small scalability because of the nature of memristive switches. By applying the IMP method, the number of memristors to make a gate can be saved. The less number of memristive switches causes less power consumption and cost. Table 2 provides the number of memristors required to make various logic gates in both cases. We note that the number of memristors to make gates in the IMP approach in Table 1 is different from Table 2. When logic operations listed in Table 1 operate on two variables A and B, the original logic would be lost during the implication process. For instance, in case of NAND, only B is changed and in case of XOR both A and B will change. Therefore, to store data we require additional number of memristive switches. For a more detailed explanation see [17]. Despite the overhead introduced in the crossbar approach in terms of number of

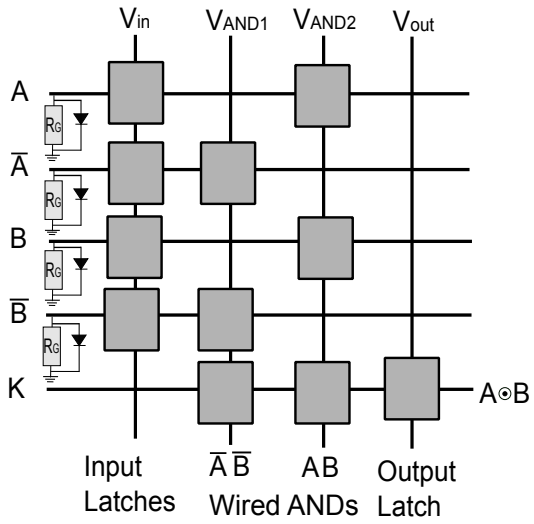

Figure 7 Crossbar array architecture for exclusive-NOR function.

Table 2 The number of memristive switches to make logic gates for the imp and crossbar array approaches.

Logic Operation	#IMP	#Crossbar Array
$S \leftarrow$ NOT A	2	3
$S \leftarrow A$ AND B	4	6
$S \leftarrow A$ NAND B	3	6
$S \leftarrow A$ OR B	6	6
$S \leftarrow A$ NOR B	6	6
$S \leftarrow A$ XOR B	7	11

memristors, it can dynamically adapt its logic function by controlling voltages. Therefore, based on the run-time requirements, the crossbar array approach is adaptable and reconfigurable.

6 Conclusions In this paper, the electrical model of the memristor is represented to show how physical properties of this device can be utilized for digital circuit applications. Memristor is a two-terminal device, therefore, we need new approaches to apply it to operate as a switch. IMP logic with memristor is studied as the first method to create digital gates. We demonstrated that by using IMP and the inverse function (NOT), it is possible to produce all digital boolean functions. In the IMP approach, the number of memristors to make gates are fewer than the crossbar array approach. We purposed the crossbar array architecture as the second novel promising technique for nanocomputing paradigm. The crossbar array method with memristive switches has been investigated comprehensively. Reconfigurability is the most significant advantage of using the crossbar array architecture. It is interesting to note that in both computing techniques, the storing and computing units are physically the same. This property has the potential to overcome the memory bottleneck. For future work, one can organize a Programmable Logic Device (PLA) platform in such a way that the crossbar array operates as a programmable AND array beside IMP as a fixed connection for the OR array.

References

[1] H. (Helen) Li and M. Hu, Compact model of memristors and its application in computing systems, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2010, (March 2010), pp. 673–678.

[2] Y. Ho, G. M. Huang, and P. Li, Nonvolatile memristor memory: device characteristics and design implications, in: Proceedings of the 2009 International Conference on Computer-Aided Design, , ICCAD '09 (ACM, New York, NY, USA, 2009), pp. 485–490.

[3] M. D. Ventra and Y. V. Pershin, CoRR **abs/1211.4487** (2012).

[4] T. Raja and S. Mourad, Digital logic implementation in memristor-based crossbars - a tutorial., in: DELTA, (IEEE Computer Society, 2010), pp. 303–309.

[5] J. Borghetti, Z. Li, J. Straznicky, X. Li, D. A. A. Ohlberg, W. Wu, and D. R. Stewart, PNAS **106** (2009).

[6] E. Lehtonen, J. Poikonen, and M. Laiho, Implication logic synthesis methods for memristors, in: Circuits and Systems (ISCAS), 2012 IEEE International Symposium on, (May 2012), pp. 2441–2444.

[7] G. S. Rose, J. Rajendran, H. Manem, R. Karri, and R. E. Pino, Proceedings of the IEEE **100**(6), 2033–2049 (2012).

[8] I. Vourkas and G. Sirakoulis, Nanotechnology, IEEE Transactions on **11**(6), 1151–1159 (2012).

[9] S. Kvatinsky, A. Kolodny, U. Weiser, and E. Friedman, Memristor-based imply logic design procedure (2011), pp. 142–147, cited By (since 1996) 4.

[10] E. Lehtonen, J. Poikonen, and M. Laiho(May), 2441–2444 (2012).

[11] E. Lehtonen and M. Laiho, Stateful implication logic with memristors, in: Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures, , NANOARCH '09 (IEEE Computer Society, Washington, DC, USA, 2009), pp. 33–36.

[12] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, Nature **464**(7290), 873–876 (2010).

[13] P. J. Kuekes, D. R. Stewart, and S. R. Williams, Journal of Applied Physics **97**(3) (2005).

[14] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, Nature **453**(7191), 80–83 (2008).

[15] L. Chua and S. M. Kang, Proceedings of the IEEE **64**(2), 209–223 (Feb. 1976).

[16] G. Snider, Applied Physics A **80**, 1165–1172 (2005).

[17] K. Bickerstaff and E. Swartzlander, Memristor-based arithmetic, in: Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on, (Nov.), pp. 1173–1177.