9-3-2021

# A 3D Crossbar Architecture for both Pipeline and Parallel Computations

John M. Acken
*Portland State University*, john.acken@pdx.edu

Muayad J. Aljafar
*Portland State University*



Citation Details

Aljafar, M. J., & Acken, J. M. (2021). A 3-D Crossbar Architecture for Both Pipeline and Parallel Computations. IEEE Transactions on Circuits and Systems I: Regular Papers, 68(11), 4456-4469.

# A 3D Crossbar Architecture for both Pipeline and Parallel Computations

Muayad J. Aljafar and John M. Acken

***Abstract*— A 3D architecture made up of a CMOS layer combined with a 3D stack of bipolar memristor crossbar arrays provides an innovative approach to hardware support for utilizing the strength of CMOS combined with the strength of memristors. Memristors have been evaluated for implementing a broad spectrum of applications such as memory, computations, hardware-based security primitives, cryptography, etc., and numerous studies have shown that memristors are desirable candidates for such applications. This paper proposes a novel 3D memristive crossbar architecture (i.e., a stack of memristive crossbar arrays built on top of CMOS substrate) with a specific focus on the way of connecting the crossbar arrays to the CMOS layer. The proposed architecture is configurable and allows restructuring crossbar arrays and creating 1D arrays with adjustable sizes. The proposed architecture enables parallel and pipeline computations where data can move or be processed in planes perpendicular to the stacked crossbar arrays. In addition, the proposed architecture is scalable meaning that stacks of crossbar arrays can be connected without additional overhead. This paper shows examples of implementing a full adder, a 4-bit look-ahead carry generator, and an 8-bit multiplexer. Simulations and area, delay, and power analysis demonstrate the behavior of the proposed 3D circuit.**

***Index Terms*—3D circuit, memristor crossbar arrays, parallel and pipeline computation, stateful logic computation**

## I. Background and Introduction

THE inherent memory wall bottleneck of the von Neumann architecture and the end of Moore's law being approached has led to a scramble for new architectures and nanodevices. One innovative architecture combines a 3D memristor crossbar array with a 3D architecture made up of a CMOS layer and memristor crossbar array [1]. Memristors [2]-[3] are examples of nanodevices. Memristors are the fourth passive element with many attractive properties—the other three elements are resistors, capacitors, and inductors. Memristors have memory, but the other three do not. Memristors are low-power devices with fast switching speed, high endurance, excellent scalability, and CMOS compatibility [4]-[8]. Memristors have been proposed for a broad spectrum of applications including, memory, computation, and hardware-based security applications such as PUFs (Physically Unclonable Functions) [9]-[11], [32]-[33]. Examples of new architectures for digital memories or reconfigurable logic circuits are CMOL (Cmos+MOLeculer-scale devices) FPGA [1], [12], FPNI (Field Programmable Nanowire Interconnect) [13], three-dimensional crossbar arrays of rectifying memristors [14], 3D crossbar arrays for network on chip based on post-silicon devices [15], and three-dimensional integration of Carbon Nanotube FETs with Silicon CMOS [16]-[17]. CMOL FPGA consists of a crossbar array fabricated above a sea of CMOS inverters. This crossbar array is slightly rotated for connecting each nanowire electrically to one metallic pin (or interconnect) extending up from the CMOS substrate. FPNI tradeoffs some speed, density, and fault-tolerance of CMOL in exchange for easier fabrication, lower power dissipation, and greater freedom in selecting nanowires in crossbar array [13]. The CMOS-memristor architectures have further improved by fabricating stacked crossbar arrays of *rectifying memristors* in an attempt to mitigate the sneak currents in crossbar arrays [14]. In addition to memristor technology, other technologies such as carbon nanotube FETs with silicon CMOS have been proposed in 3D architectures [16]-[17].

This paper proposes a novel 3D memristive crossbar architecture. The novelty of this architecture is in the way of connecting the crossbar arrays to the CMOS layer. There are existing 3D CMOS-memristor architectures as described in the previous paragraph. The emphasis in our paper is not speed, power, or density but rather the flexibility of the circuits and their interconnections. The proposed architecture provides programmable flexibility in connecting the stacked crossbar arrays. These connections are configurable and can turn the 3D architecture into, e.g., 1D arrays with adjustable sizes for parallel and pipeline computing, resulting in a highly configurable architecture. In addition, the proposed architecture can move or process data in planes perpendicular to the stacked crossbar arrays. This circuit architecture is different from conventional 3D memristive architectures in the literature. The proposed architecture also provides programmable flexibility for implementing completely different functions such as RAM arrays, logic gates, etc. This paper shows examples of implementing some combinational circuits (*n*-bit adder and 8-bit multiplexer), demonstrating the behavior of the proposed architecture. The rest of the paper is organized as follows. Section II describes the new 3D crossbar architecture, and

Manuscript received December 18, 2020; revised April 5, 2021; revised June 19, 2021. Muayad J. Aljafar is with the Information and Communication Engineering Department, Basra College, Basra, Iraq (e-mail: m.aljafar@gmail.com).

John M. Acken is with the Electrical and Computer Engineering Department, Portland State University, Portland, OR 97201 USA (acken@pdx.edu).

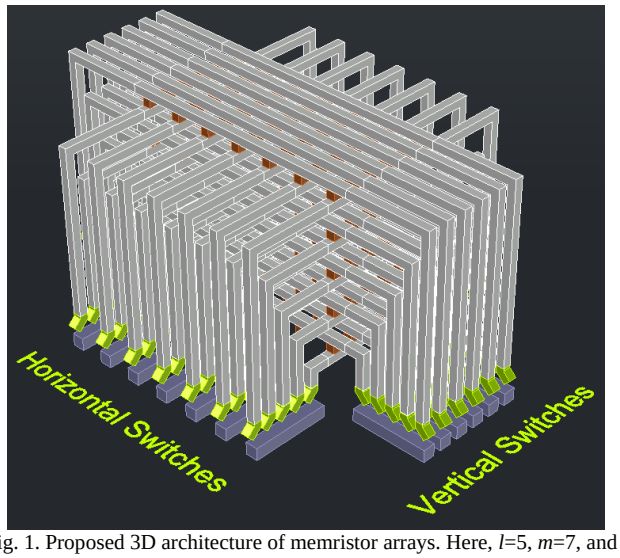


Fig. 1. Proposed 3D architecture of memristor arrays. Here, $l$=5, $m$=7, and $n$=6.

Section III shows examples of implementing circuits in the new architecture. Section IV evaluates the proposed architecture, and Section V concludes the paper.

## II. Proposed Implementation

The proposed architecture consists of $l$ (limited by technology) stacked memristive crossbar arrays of $m \times n$ and allows processing data in any plane (XY, YZ, and XZ) in the 3D architecture. This unconventional 3D architecture requires different interconnects (for connecting a CMOS chip to crossbar nanowires) from ones introduced in the CMOL architecture [1]. The proposed architecture utilizes only two-terminal selectors, i.e., memristors with *intrinsic current-rectifying characteristic*. This current-rectifying characteristic can 1) effectively break the *sneak current paths* and is a key reason that the array proposed here can operate without having an external transistor or diode at each crosspoint, 2) reduce the difficulties of fine-tune programming of memristors in crossbar arrays, and 3) enable multi-bit storage ability [19]. The proposed architecture enables in-memory computations and overcomes the memory bottleneck in von-Neumann architectures. Details of the circuit architecture are available in Section II-A.

### A. Architecture Details

Fig. 1 shows a schematic of the proposed architecture. The crossbar arrays are stacked above a CMOS layer in XY planes extending up toward Z-axis. The crossbars and CMOS circuits communicate through nanowires and CMOS switches shown in Fig. 1. The CMOS layer supplies the voltage needed for operating the crossbar arrays where logic and storage are implemented. For the following description, we use five stacked 7×6 crossbar arrays. As mentioned earlier, data can move in any plane perpendicular to the stacked crossbar arrays, and for that, the CMOS switches play a major role. Fig. 2 shows how horizontal CMOS switches can connect rows of (seven) memristors in an XZ plane. The brown squares are memristors located at intersections of gray nanowires. Yellow planes are insulating layers. The CMOS peripheral switches (on both

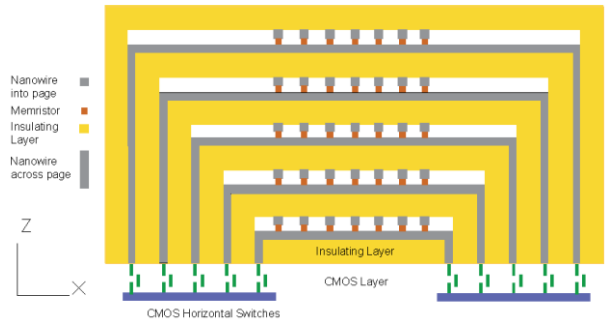


Fig. 2. Schematic of the proposed circuit in an XZ plane.

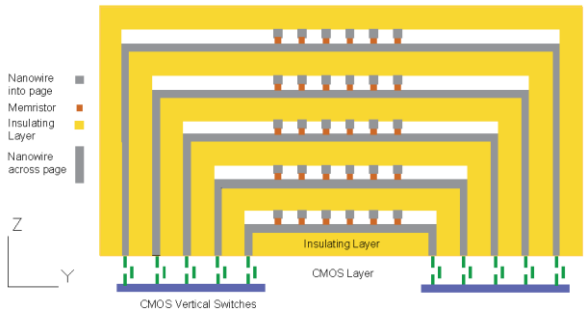


Fig. 3. Schematic of the proposed circuit in an YZ plane.

sides) allow data to move between the stacked crossbar arrays in XZ planes. Each XZ plane can be used for implementing multiple operations in parallel. For example, connecting two rows of memristors by two CMOS switches results in three separate arrays, i.e., one with a size of 1×14 and the others with a size of 1×7. These three arrays can implement three different types of operations all at once. Note that the horizontal and vertical switches connect the relative memristors anti-serially. Fig. 3 shows the stacked crossbar arrays in a YZ plane. Note that gray lines are vertical nanowires connected to vertical switches. These switches allow data to move between stacked crossbar arrays in YZ planes. Since there are five stacked crossbar arrays, each plane of memristors (seen as a row or column of memristors in Fig. 2 and Fig. 3) is connected to five CMOS switches from each side. When all vertical and horizontal switches are turned off, the stacked crossbar arrays are disconnected, and each crossbar array can implement only one type of logic operation [18]. Therefore, adding vertical and horizontal switches facilitates reconfiguring the topology of the architecture for compact and efficient use. The circuit in Fig. 4 shows the functionality of the peripheral CMOS switches. Each nanowire (a gray line in Fig. 1) on each side is connected to a non-inverting tristate buffer and a transmission gate (Fig. 4a). The crossbar arrays communicate through these switches. For example, moving one bit from crossbar A1 to crossbar B1 occurs through a non-inverting tristate buffer connected to a crossbar A1 nanowire, and a transmission gate connected to a crossbar B1 nanowire (Fig. 4b). The non-inverting tristate buffer is used as an amplifier, whereas a transmission gate is used as a selector. The overall number of transistors used in Fig. 4a is 10 including two transistors (an inverter) used for generating $\overline{En}$. These peripheral switches are fabricated in the CMOS layer. There are more transistors required to select a connection to multiple voltage levels as discussed in Section IV.

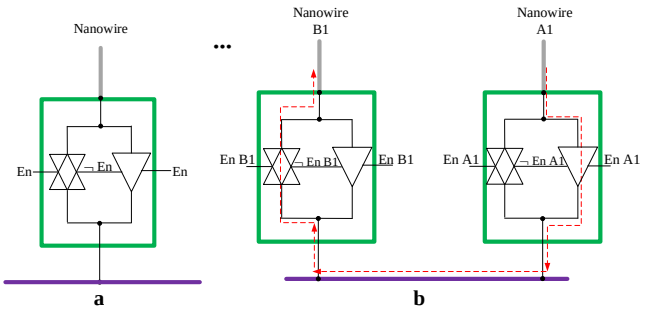


Fig. 4. (a) Schematic of peripheral CMOS switches connected to one side of each crossbar nanowire in Fig. 1. (b) Example of connecting two nanowires of two crossbar arrays.

### B. *Fabrication Possibilities*

Improvements in fabrication technology for 3D memristors demonstrate the feasibility of our architecture. Although memristor fabrication is improving there are still several challenges ahead such as heat dissipation, routing difficulties for active (1T1R) arrays, and high-temperature steps. An example of the feasibility of our architecture is provided by Li *et al*. [14] where they practically demonstrated several stacked crossbar arrays of rectifying memristors. They fabricated three stacked crossbar arrays using the fluid-supported membrane transfer technique. They used a hydrogen silsesquioxane layer between each layer of a crossbar array for electrical isolation. In addition, the authors fabricated five stacked crossbar arrays using a different approach. They showed successful readout for different scenarios confirming the efficient blocking of sneak paths by the rectifying memristors. Furthermore, the authors simulated the readout resistance in different states in a 64×64 crossbar array with different layers, considering the worst-case scenario. Their simulations also show the correct operation of the 3D circuit. These examples in the literature show the possibility of fabricating 3D crossbar arrays similar to our proposed one.

## III. Example Applications

This section shows examples of calculating logic functions in the proposed 3D architecture. Here, the goal is to demonstrate the scalability of the design and show examples of both parallel and pipeline computations in the 3D architecture. The details of the following examples are included to clearly demonstrate the flexibility of this architecture. The examples are 1-bit full adder (Section III-B), 4-bit full adder (Section III-C), and 8-bit multiplexer (Section III-D) implemented with volistor and programmable diode gates [18], [24], [27] in crossbar arrays of rectifying memristors [14], [19]-[20]. The functionality of these logic gates is briefly explained in Section III-A.

### A. *Volistor and Programmable Diode Gates*

The implementation of volistor and programmable diode gates relies on rectifying memristors, i.e., memristors with *intrinsic* diode behavior. The use of rectifying memristors eliminates the sneak currents in a crossbar array due to the diode-like behavior and the need for individual selectors for each memristor. This work exploits the high rectifying ratio of the memristors for logic applications. Below, we review different implementations of some volistor and programmable diode gates utilized in the examples.

*Programmable Diode OR and Volistor NOR Gates*: Here, we show schematics for implementing an *n*-input programmable diode-OR and volistor NOR gates. Fig. 5a shows a schematic of a programmable diode OR gate, where inputs are voltage signals applied to terminal A of the memristors (see the inset in Fig. 5), and the output is the voltage on wired-OR *l*, $v_l$. When inputs are similar, either low or high, $v_l$ will be equal to the input voltages. However, when inputs are different, $v_l$ will be equal to the high input voltage. Specifically, all memristors connected to a low input are reverse-biased, expressing a high resistance as if they were programmed to an HRS. Here, the low input is chosen 0V, which encodes logic 0, and the high input is chosen 0.6V, which encodes logic 1. A configuration example under which the gate operates correctly is also shown in Fig. 5a where inputs $v_i$ are chosen to be non-destructive to the memristors' states, i.e., $v_i < V^+_{TH}$ and $V^+_{TH}$ is a memristor positive threshold voltage. The state of each memristor in Fig. 5a is used to either connect or disconnect an input to the gate. If a memristor is set (or programmed to a low resistance state, LRS), it connects the input to the gate, but if the memristor is reset (or programmed to a high resistance state, HRS), it suppresses the input. The truth table in Fig. 5 summarizes the correct behavior of the programmable diode-OR gate.

The output of the programmable diode OR gate can be stored as resistance. Fig. 5b shows an implementation of a volistor NOR gate, which stores the logic output of the diode OR in an output memristor. The output memristor is connected to bias voltage $V^-_W$, chosen to satisfy $V^-_W - v_l \leq V_{RESET}$ when at least one of the inputs connected to the gate (through a memristor in an LRS) is high; here, $V_{RESET}$ is a voltage that programs a memristor in an LRS to an HRS. A voltage configuration under which the volistor NOR gate operates correctly is also shown in Fig. 5b. This implementation requires initializing the output memristor to an LRS.

Fig. 5c and 5d show two schematics for implementing a NOR gate, where both input and output are stored in memristors (as logic resistance). These schematics, which have been used in the literature, have different implementations than the volistor NOR gate. Specifically, in volistor NOR, the wired-NOR is floated; however, the wired-NOR in other schematics is grounded through a reference resistor. (This is true for other volistor gates, as well). In Fig. 5c, $V^+_r$, a positive voltage smaller than $V^+_{TH}$, is applied to terminal A of the input memristors, $V^-_W$ is applied to the output memristor, and wired-OR *l* is grounded through reference resistor *R*g. The value of *R*g should be chosen with care to reflect the logic values stored in the input memristors. In general, *R*g should be much larger than the LRS and much smaller than the HRS. One choice for *R*g, chosen in this paper, is the geometric mean of LRS and HRS. Here, we discuss the conditions under which the NOR gate operates correctly by considering all input combinations of the gate. If at

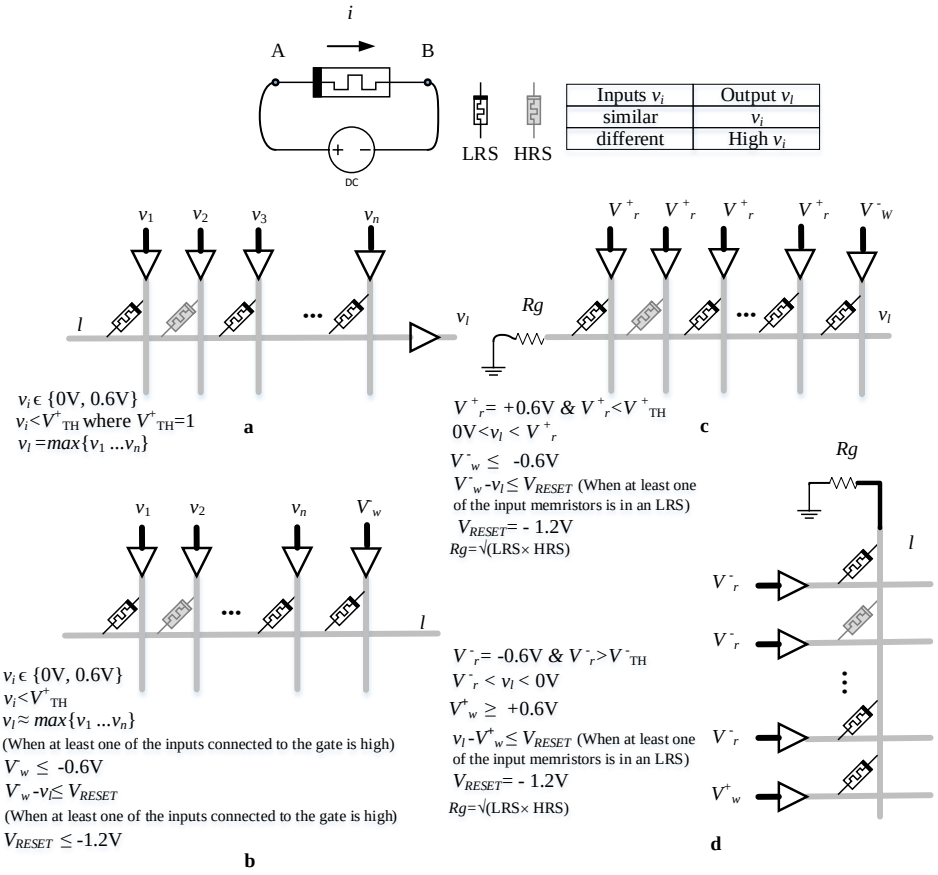


Fig. 5. Implementations of logic OR and NOR gates. (a) Schematic of a programmable diode OR gate, (b) Schematic of a volistor NOR gate, (c) Schematic of a NOR gate, (d) Schematic of a NOR gate. The inset shows the polarity of a memristor connected to a voltage source. The flow of current into the device decreases its resistance. The other inset shows symbols of a memristor in an LRS and HRS. The truth table summarizes the correct behavior of a programmable diode OR gate. Descriptions close to each schematic are examples of bias conditions under which the gates operate correctly.

least one input is logic '1', i.e., one input memristor is in an LRS, the voltage on wire-OR $l$ should be close to $V^+_r$; and therefore, the voltage across the output memristor should be about $V^-_W$- $V^+_r$. This negative voltage should be equal to or smaller than $V_{RESET}$ to program the output memristor, which is initialized to an LRS, to an HRS or logic '0'. And, if all memristors are in an HRS, the voltage on wired-OR should be close to 0V; therefore, the voltage across the output memristor should be about $V^-_W$, which is insufficient to program the memristor to an HRS. This correct behavior of the logic NOR gate can be achieved by the voltage configuration shown in Fig. 5c. The schematic of the NOR gate in Fig. 5d is similar to that in Fig. 5c. Here, a negative bias voltage, $V^-_r$, which is larger than $V^-_{TH}$, a memristor's negative threshold voltage, is applied to terminal B of the input memristors, and $V^+_W$, a positive voltage, is applied to the output memristor. A voltage configuration under which the NOR gate operates correctly is also shown in Fig. 5d. Note that the schematic of NOR can be modified for implementing a NOT gate by reducing the number of inputs to one. In this implementation, the same voltage configuration of NOR gates would realize a NOT gate. All the implementations discussed above have been used in the examples shown in Section III.

*Programmable Diode AND and Volistor AND Gates*: Here we show schematics for implementing an $n$-input programmable diode-AND and volistor AND gates. Fig. 6a shows a schematic of a programmable diode AND gate, where inputs are voltage signals applied to terminal B of the memristors (see the inset in Fig. 5), and the output is the voltage on wired-AND $l$, $v_l$. When inputs are similar, either low or high, $v_l$ will be equal to the input voltages. However, when inputs are different, $v_l$ will be equal to the low input voltage. All memristors connected to high inputs are reverse-biased, expressing a high resistance as if they were programmed to an HRS. Here, the low input is chosen 0V, which encodes logic 0, and the high input is chosen 0.6V, which encodes logic 1. A voltage configuration under which the gate operates correctly is also shown in Fig. 6a. Specifically, inputs $v_i$ are chosen to be non-destructive to the states of the memristors, i.e., $v_l - v_i > V^-_{TH}$. The state of each memristor in Fig. 6a is used to either connect or disconnect an input to the gate. If a memristor is set, it connects the input to the gate, but if the memristor is reset, it suppresses the input.

Fig. 6b shows another schematic for implementing an $n$-input programmable diode AND gate. Here, inputs are resistance, whereas the output is voltage. Specifically, inputs are states of memristors connected to reference resistors, $Rg_i$. A positive

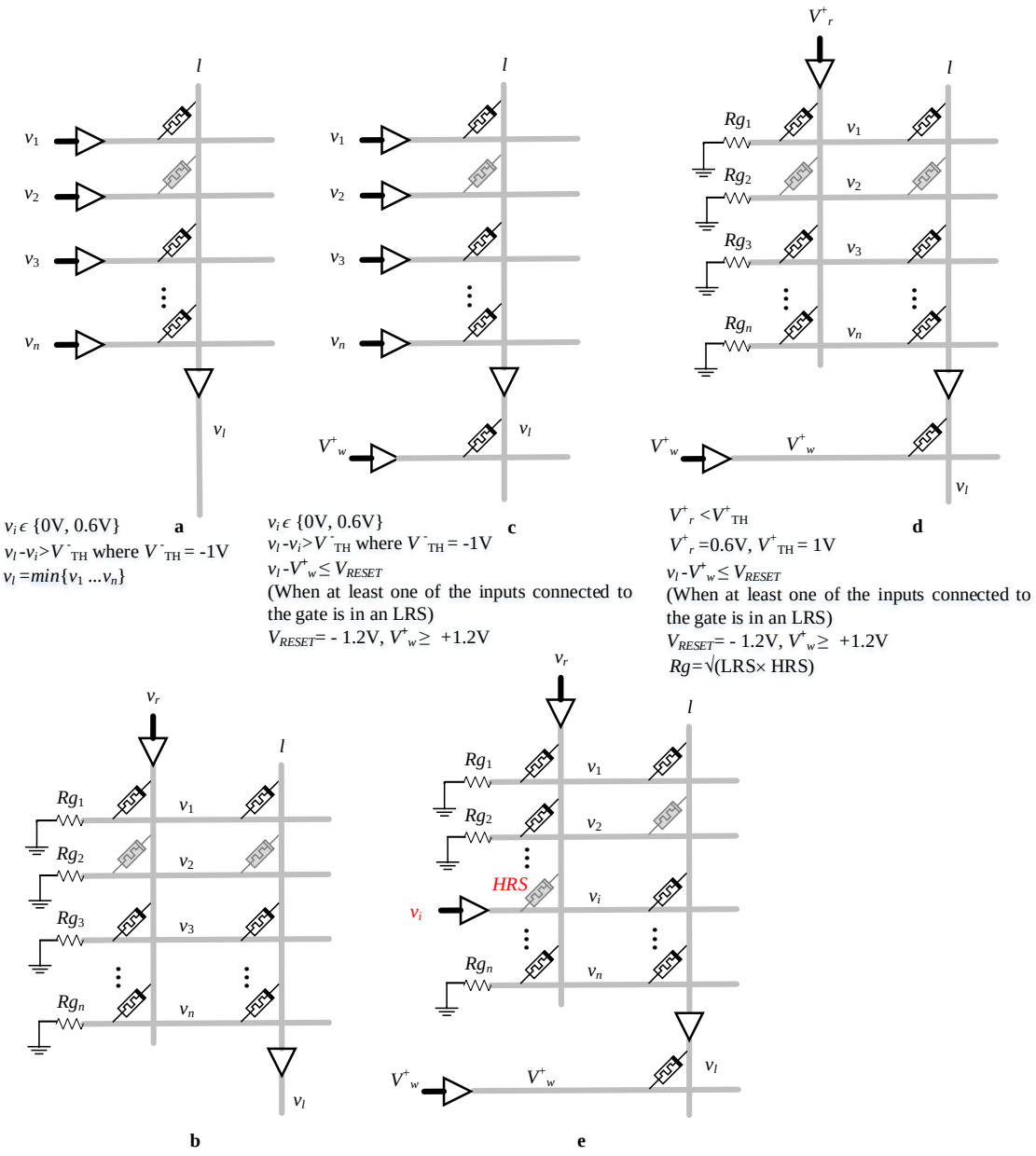


Fig. 6. Implementations of a logic AND gate. (a) Schematic of a diode AND gate where both inputs and the output are voltage. (b) Schematic of a diode AND gate where inputs are resistance, and the output is voltage. (c) Schematic of a volistor AND gate where inputs are voltage and the output is resistance. (d) Schematic of a volistor AND gate where both inputs and the output are resistance. (e) Schematic of a diode AND gate where inputs are a combination of voltage and resistance, and the output can be either voltage or resistance.

voltage, $V^+_r$, smaller than $V^+_{TH}$, is applied to the input memristors to read their logic values, $v_i$, which in turn are applied to the programmable diode AND gate in Fig. 6a.

Fig. 6c shows a schematic for implementing a volistor AND gate, where inputs are voltage, and the output is resistance. In this schematic, the output of the programmable diode AND gate, shown in Fig. 6a, is stored in an output memristor, connected to bias voltage $V^+_W$. This implementation requires initializing the output memristor to an LRS. A voltage configuration under which this gate operates correctly is also shown in Fig. 6c.

Fig. 6d shows another schematic for implementing a volistor AND gate, where both input and output are resistances. In this schematic, $V^+_r$ is applied to the input memristors to read their values, $v_i$, which in turn are applied to the volistor gate, shown in Fig. 6c. This implementation requires initializing the output memristor to an LRS. A voltage configuration under which this gate operates correctly is also shown in Fig. 6d.

Fig. 6e shows the last schematic for implementing an AND gate, where inputs are a combination of voltage and resistance. This schematic is more flexible than those in Fig. 6a-Fig. 6d. The gate's output can be either voltage or resistance. Note that the memristors located at intersections of input voltages and

TABLE I
LIST OF LOGIC GATES IMPLEMENTATIONS USED IN THE EXAMPLES

| # | Logic Gates | Gate's Input-Output | Ref. |
|---|---|---|---|
| 1 | Programmable diode OR | V-V | Fig. 5a |
| 2 | Volistor NOR | V-R | Fig. 5b |
| 3 | NOR | R-R | Fig. 5c |
| 4 | NOR | R-R | Fig. 5d |
| 5 | Programmable diode AND | V-V | Fig. 6a |
| 6 | Programmable diode AND | R-V | Fig. 6b |
| 7 | Volistor AND | V-R | Fig. 6c |
| 8 | Volistor AND | R-R | Fig. 6d |
| 9 | AND | {V, R}-{V} or {R} | Fig. 6e |

input resistances must be programmed to an HRS, as shown in Fig. 6e. For ease of reference, all schematics discussed above are summarized in Table I.

One advantage of using volistor and programmable diode gates is that the bias voltages connected to output memristors will not disturb the states of other memristors connected to the input voltages. The reason is that the output memristors are always reverse-biased and suppress the reverse currents. Let's consider the NOR gate in Fig. 5b, as an example. For any input combinations, the voltage across the output memristor is negative, therefore the memristor is always reverse-biased and suppresses the current. In addition, the voltage across other memristors is sufficiently smaller than $V^{+}_{TH}$=1V or larger than $V^{-}_{TH}$=-1V and does not disturb their resistance states. For example, when the inputs are equal, the voltage across these memristors is about 0V, or when the inputs are different, the voltage is close to either 0V or $V^{-}_{TH}/2$, which induces insignificant state drifts in these memristors.

### *B. 1-bit Full Adder*

There are serval ways of implementing arithmetic adders in crossbar memristors, e.g., [21]-[26], [30]. This paper cascades programmable diode OR gates [24] (i.e., gate 1 in Table I, implemented in crossbar A in Fig. 7) and volistor AND gates (i.e., gate 7 in Table I, implemented in crossbars B and C in Fig. 7) for realizing a 1-bit full adder. The top two arrays are preprogrammed to be 1-bit adder. Red memristors in the lower array contain the results based upon voltage inputs. Inputs and their complements are applied to crossbar array A1 as voltage signals, i.e., $v_a$, $v_b$, $v_c$, $v_{a'}$, $v_{b'}$, and $v_{c'}$. If $v_a$ is a high input voltage, its complement $v_{a'}$ is a low voltage input and vice versa. The adder calculates the outputs ($s$ and $c_o$) in one clock cycle. Crossbar arrays A1 and B1 are connected through horizontal CMOS switches, whereas crossbar arrays B1 and C1 are connected through vertical CMOS switches—here, only two columns of crossbar arrays B1 and C1 are connected. The adder is implemented in the POS (Product-Of-Sums) form where sums are realized by wired-OR logic in rows of crossbar array A1, and products are realized by wired-AND logic in columns of crossbar array B1. For example, the voltage on the top most nanowire in crossbars A and B encodes logic $a+b+c$, and the voltage on the left most nanowire in crossbars B1 and C1 encodes logic $(a+b+c)(a+b'+c')(a'+b'+c)(a'+b+c')$. $V_{SET}$ in

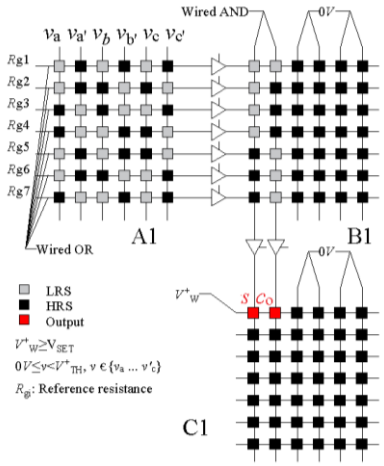


Fig. 7. Implementation of a 1-bit full adder in the proposed 3D architecture. The top two arrays are preprogrammed. The adder input is the voltages at the top. The red memristors contain the results.

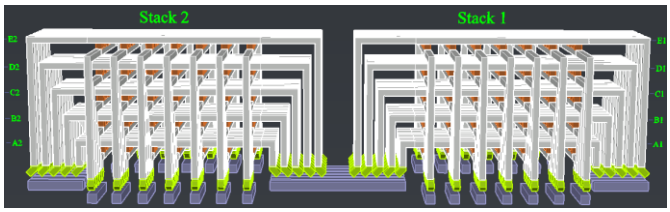


Fig. 8. Connecting two stacks of the proposed 3D architecture. The crossbar arrays in Stack 1 and Stack 2 are labeled A1, B1, etc. for referencing.

Fig. 7 is a positive voltage that toggles an HRS of a memristor to an LRS, $V^{+}_W$ ($\geq V_{SET}$) is a positive voltage applied to the output memristor in volistor AND gate, and $R_g i$ is a reference resistor, which connects a nanowire to the ground. This implementation does not change the states of memristors in crossbar arrays A1 and B1, hence, the same crossbar arrays can implement a $k$-bit ripple carry adder in about $k$ clock cycles. Black squares are memristors in an HRS, and gray squares are memristors in an LRS. This example shows how peripheral switches can customize the crossbar structure for pipeline computing.

### *C. 4-bit Look-ahead Carry Generator*

This example utilizes two stacks of the crossbar arrays (Stack 1 and Stack 2 in Fig. 8) connected through horizontal CMOS switches. Crossbar arrays A1, B1, C1, D1, and E1 belong to Stack 1, and crossbar arrays A2, B2, C2, D2, and E2 belong to Stack 2. Similarly, a network of the proposed architecture can be connected through the vertical and horizontal switches. The carry generator is implemented in three steps by, first, calculating complements of carry bits, second, calculating the carry bits, and third, calculating the sum bits. The circuit configuration in Fig. 9a shows how to calculate complemented carry bits $c'_1$, $c'_2$, and $c'_3$ by cascading AND-NOR gates. This step is implemented in one clock cycle. Inputs are $a$, $b$, and $c_{in}$

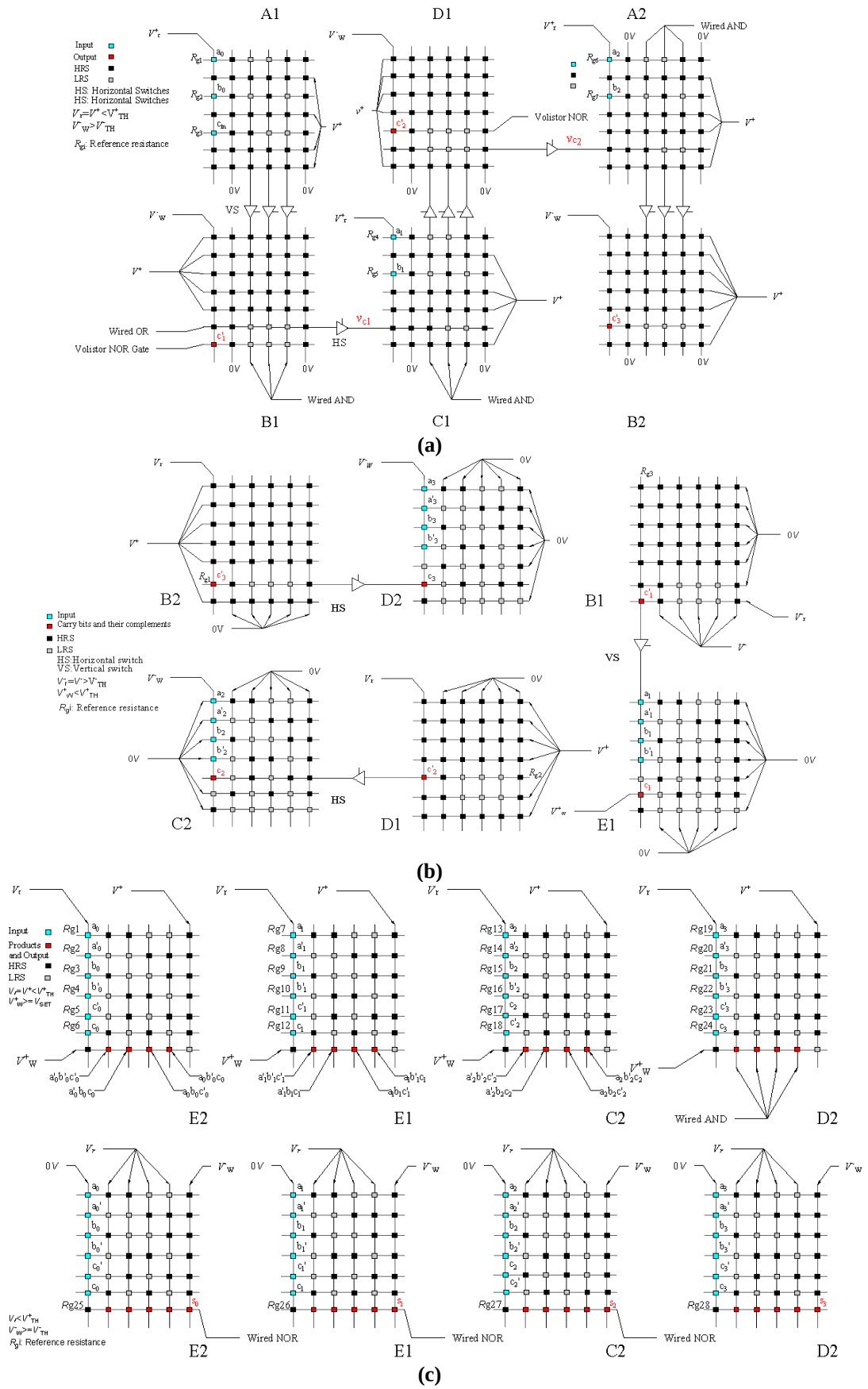


Fig. 9. Implementation of 4-bit look-ahead carry generator. (a) Computing $c'_1$, $c'_2$, and $c'_3$, complements of the carry bits. (b) Computing the carry bits and their complements. (c) Calculating the sum bits $s_0$, $s_1$, $s_2$, and $s_3$.

stored in cyan memristors, and outputs are $c'_1$, $c'_2$, and $c'_3$ stored

in red memristors. The complemented carry bits, $c'_1$, $c'_2$, and $c'_3$, are calculated in the SOP (Sum-of-Products) form. Specifically, $c'_1$ is calculated by cascading three programmable diode AND gates (i.e., gate #6 in Table I) in crossbar A1, and a 3-input volistor NOR gate (i.e., gate #2 in Table I) in crossbar B1. The products computed in crossbar A1 are $a_0b_0$, $b_0c_{in}$, and $a_0c_{in}$. $c'_2$ is calculated by cascading three AND gates (i.e., gate #9 in Table I) in crossbar C1 and a 3-input volistor NOR gate (gate #2 in in Table I) in crossbar D1. The products computed in crossbar C1 are $a_1b_1$, $b_1c_1$, and $a_1c_1$. The input voltage to the AND gates, $v_{c1}$, which encodes carry bit $c_1$, is calculated in crossbar B1 by a programmable diode OR gate (i.e., gate #1 in Table I). Similarly, $c'_3$ is calculated by cascading three AND gates (gate #9 in Table I) in crossbar A2 and a 3-input volistor NOR gate (gate #2 in Table I) in crossbar B2. The products computed in crossbar A2 are $a_2b_2$, $b_2c_2$, and $a_2c_2$. The input voltage to the AND gates, $v_{c2}$, which encodes carry bit $c_2$, is calculated in Crossbar D1. In Fig. 9a, $V^+$ (which is equal to $V^+_r$, a positive voltage for reading a state of a memristor) is applied for suppressing sneak currents. Next, carry bits $c_1$, $c_2$, and $c_3$ are calculated and stored in crossbar arrays E1, C2, and D2, respectively, by using three NOT gates (Fig. 9b). This step is realized in one clock cycle. Note that the NOT gate that calculates $c_1$ (in crossbar B1 and E1) uses the same voltage configuration as the NOR gate in Fig. 5d, whereas the other two NOT gates that calculate $c_2$ and $c_3$ (in crossbar D1 and C2 and in crossbar B2 and D2) use the same voltage configuration as the NOR gate in Fig. 5c. In Fig. 9b, $V^-$ (which is equal to $V^-_r$, a negative voltage for reading a state of a memristor) is applied for suppressing sneak currents. Next, complements of these carry bits are again calculated (by using three NOT gates with the same voltage configuration as in the NOR gate in Fig. 5c) and stored in the *same crossbar arrays* where carry bits are located. This step, which is not shown in Fig. 9, is realized one clock cycle. Finally, sum bits $s_0$, $s_1$, $s_2$, and $s_3$ are calculated in the SOP form and stored in the bottom-right memristors in crossbar arrays E2, E1, C2 and D2 in Fig. 9c. This step is implemented in two clock cycles. In the first clock cycle, products $a'_i b'_i c'_i$, $a'_i b_i c_i$, $a_i b_i c'_i$ and $a_i b'_i c_i$ are calculated simultaneously with volistor AND gates (i.e., gate #8 in Table I), as shown in the upper crossbar arrays in Fig. 9c. And in the second clock cycle, the sums of the products (sum bits) are calculated with NOR gates (gate #3 in Table I), as shown in the lower crossbar arrays in Fig. 9c. Note that $V^+_W$ ($>=V_{SET}$) connected to the output memristors in volistor AND gates (Fig. 9c) is different from $V^+_W$ ($<V^+_{TH}$) connected to the output memristors in NOT gates (Fig. 9b). In all computations, the working memristors (i.e., the gray and black memristors) maintain their resistance states. Therefore, only input and output memristors (i.e., the cyan and red memristors) need to be reprogrammed for implementing the next 4-bit addition. Programming the input and output memristors takes only four clock cycles. Overall, nine clock cycles are required for implementing the 4-bit look-ahead carry generator as explained above.

### *D. 8-bit Multiplexer*

This example shows an 8-bit multiplexer implementation in stacked crossbar arrays A1, B1, C1, D1, and E1. The multiplexer selects between inputs $a$ and $b$ based on the selector's value, $x$ (i.e., $c=x'a+xb$ where $c$ is the multiplexers' output). This example utilizes AND and NOR gates and implements the multiplexer in $c=(x'a'+xb')'$ form, which requires writing the input complements (instead of true inputs) in the input memristors. Fig. 10 shows crossbar arrays for calculating $x'a$ (Fig. 10a and Fig. 10b) and $x'a+xb$ (Fig. 10c and Fig. 7d), and the crossbar arrays for calculating $xb$ are not shown, as they are very similar to Fig. 10a and Fig. 10b. Cyan memristors show values of $a$ and $b$, and orange memristors show the value of $x$ and its complement, $x'$. The circuit calculates $x'a$ (and $xb$) with volistor AND gates (i.e., gate #7 in Table I) and stores the results in red memristors. These products are calculated in two consecutive clock cycles due to the limited size of crossbar arrays. In the first clock cycle, input bits $a_4$ … $a_0$ (and $b_4$ … $b_0$) stored in crossbar A1 (and C1) are applied to crossbar B1 in Fig. 10a (and D1) for calculating products $x'a_4$, $x'a_3$, $x'a_2$, $x'a_1$, $x'a_0$ (and $xb_4$, $xb_3$, $xb_2$, $xb_1$, $xb_0$). And, in the second clock cycle, input bits $a_7$ $a_6$ $a_5$ (and $b_7$ $b_6$ $b_5$) stored in crossbar B1 (and D1) are applied to crossbar A1 in Fig. 10b (and C1) for calculating products $x'a_7$, $x'a_6$, $x'a_5$ (and $xb_7$, $xb_6$, $xb_5$). Voltage values $V_{a7x'}$, $V_{a6x'}$… $V_{a0x'}$ on vertical nanowires in Fig. 10a and Fig. 10b show products values $a_7x'$, $a_6x'$… $a_0x'$. Next, the circuit NORs the products and stores the results (the multiplexer's output) in crossbar array E1. Due to the limited size of crossbar arrays, the NOR step is also implemented in two clock cycles. In one clock cycle, NOR operations $c_0=(x'a_0+xb_0)'$, $c_1=(x'a_1+xb_1)'$, $c_2=(x'a_2+xb_2)'$, $c_3=(x'a_3+xb_3)'$, and $c_4=(x'a_4+xb_4)'$ are realized (Fig. 10c), and in the second clock cycle, NOR operations $c_5=(x'a_5+ xb_5)'$, $c_6=(x'a_6+xb_6)'$, and $c_7=(x'a_7+xb_7)'$ are performed (Fig. 10d). These NOR operations are implemented with NOR gate #4 in Table I. Overall, the implementation of an 8-bit multiplexer takes eight cycles. In the first clock cycle, the input and output memristors are programmed to an HRS. In the second and third clock cycles, the output memristors are programmed to an LRS, and in the fourth clock cycle, the input memristors are programmed by the *input complements*. In the next four clock cycles, the 8-bit multiplexer is implemented.

## IV. Area, Time, and Power Evaluations

This section analyzes the size, delay, and power of the proposed architecture. The size was evaluated with respect to the number of transistors and memristors, the delay was measured based on the number of clock cycles for implementing a function, and the mean power of all circuits discussed above was calculated using the LTspice simulator, as well. There are five transistors required for selecting a connection to one of four voltage levels or a reference resistor. These transistors are different from the peripheral switches shown in Fig. 4. Each nanowire in the crossbar arrays can be driven by four voltage levels $V^-_r$ ($>V^-_{TH}$), $0V$, $V^+_r$ ($< V^+_{TH}$), and $V^+_W$ ($> V_{SET}$), connected to reference resistor $R_g$, and set to high

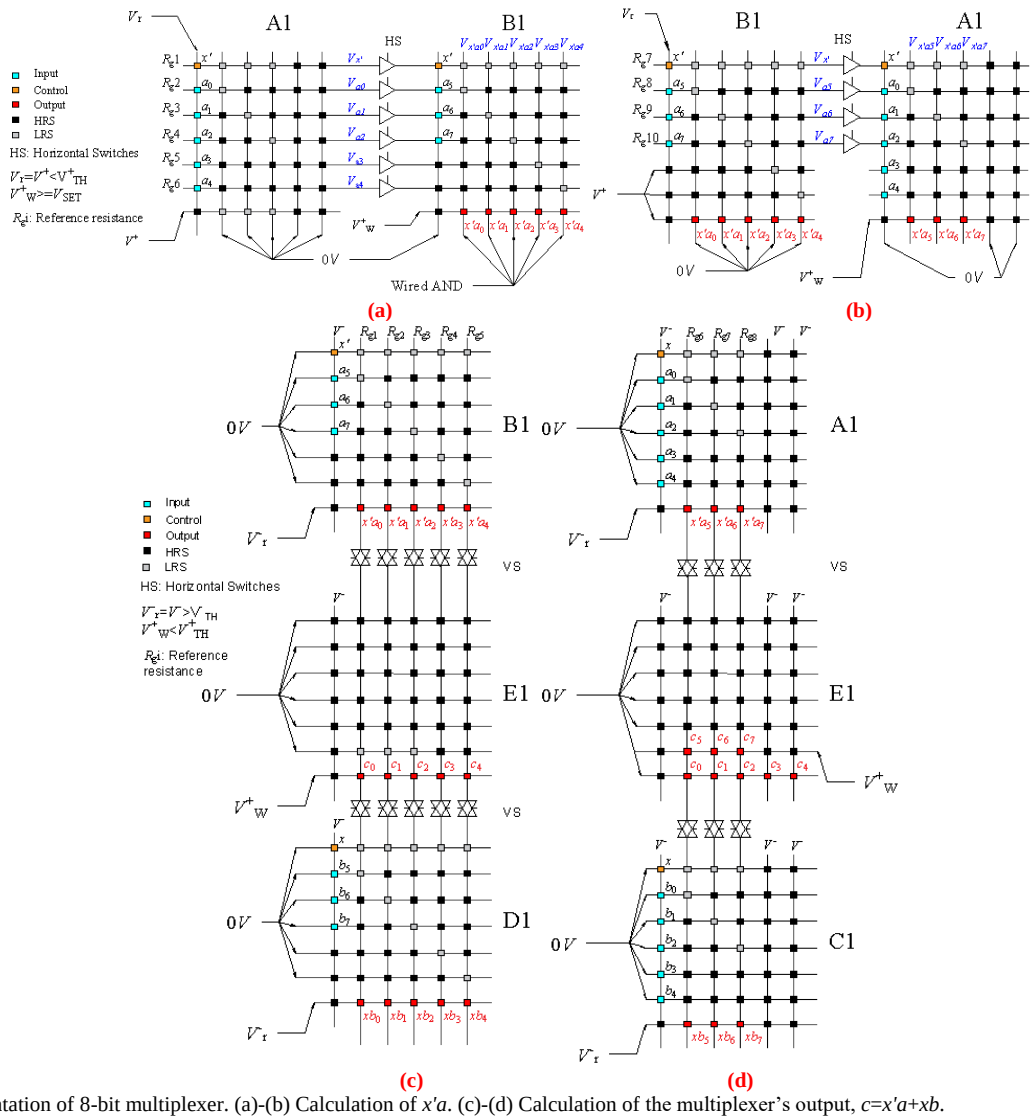


Fig. 10. Implementation of 8-bit multiplexer. (a)-(b) Calculation of $x'a$. (c)-(d) Calculation of the multiplexer's output, $c=x'a+xb$.

impedance $Z$, where $V^-_r$ and $V^+_r$ are non-destructive voltages to the states of memristors and are chosen to satisfy $V^-_r - V^+_r \leq V_{RESET}$. These voltage levels are sufficient for both logic computation and crossbar initialization, and the same CMOS circuits implement both operations. Setting and resetting operations require connecting a memristor to $V_{SET}$ and $V_{RESET}$, respectively, and these operations are realized in the same manner as in a 2D crossbar array. In other words, there is no difference between selecting a specific memristor in our 3D architecture and a 2D crossbar array.

TABLE II
MEMRISTOR PARAMETERS AS USED IN [28]

| Memristor Parameter | Value |
|---|---|
| $V_{RESET}$ | -1.2$V$ |
| $V^-_{TH}$ | -1$V$ |
| $V^+_{TH}$ | 1$V$ |
| $V_{SET}$ | 1.2$V$ |
| $R_{ON}$ | 500 KΩ |
| $R_{OFF}$ | 500MΩ |
| $T$ | 4ns |
| α | $125\times10^7$ $(Vs)^{-1}$ |

### A. *Simulations*

The proposed architecture was simulated for implementing all circuits explained above using the LTSpice simulator, 50nm TSMC process BSIM4 models, and a rectifying memristor model used in [28]. Table II shows the memristor parameters. $R_{ON}$ and $R_{OFF}$ are the low and high resistance values of a rectifying memristor, $T$ is the state transition delay of a memristor, and α is a positive constant related to the programming rate. For simplicity, we assume that $V^-_{TH}$ and $V^+_{TH}$ are symmetric, but in practice, these threshold voltages may not be symmetric. The goal here is to investigate the proposed architecture with a multitude of memristors, and this simple model allows for efficient simulations. Therefore, the simulation results should be considered only as suggestive as the physical implementations might show different characteristics. We use the same reasoning as in [28] to emphasize architectural behavior rather than detailed circuit behavior. This goal can be achieved even though this model is a simple one. For example, the model assumes a piecewise linear dependency between its programming rate and the

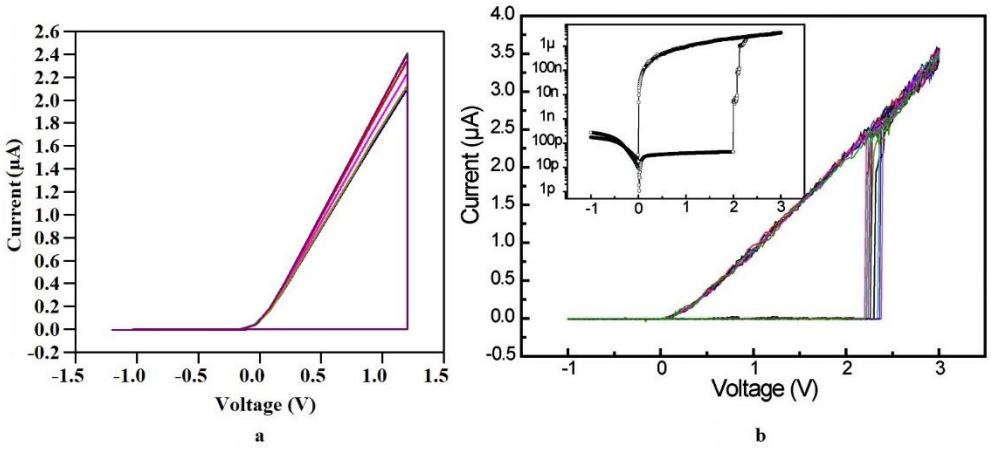


Fig. 11. (a) I-V characteristics of the rectifying memristor model used in this work [28]. (b) I-V characteristics of fabricated rectifying memristors shown in [19].

applied voltage. It also assumes fixed threshold voltages, which is not correct for many physical devices [19]-[20].

The memristor model used assumes a diode-like behavior only at LRS similar to [29]. In other words, the device is non-rectifying at HRS, but it shows a high resistance value. However, in [16], the rectifying memristors in an HRS are more conductive when forward-biased compared to when reverse-biased. Fig. 11a shows the I-V characteristics of the rectifying memristor model used [28] with varied α sampled 10 times from a normal distribution with mean $125\times10^{7}$ $(Vs)^{-1}$ and standard deviation $50\times10^{6}$ $(Vs)^{-1}$. Fig. 11b shows the I-V characteristics of 10 different rectifying memristors from a fabricated crossbar array with 10 Gbits/cm$^2$ density [19]. The model used simulates the behavior of the real memristors when reverse-biased.

If a memristor model with a diode-behavior (for both HRS and LRS) is used in our simulations, the circuits shown above work more efficiently by further reducing the sneak path currents in crossbar arrays and improving logic implementations in crossbar arrays by increasing the margin between logic states. For example, in the volistor NOR gate shown in Fig. 5b, the memristor in an HRS allows a smaller reverse current compared to our Spice implementation, in which the reverse current is equal to the forward current. Our Spice implementation would show a smaller voltage on wired-NOR when the number of reverse-biased memristors in an HRS increases. This smaller voltage would increase the transition delay in the output memristor and the power dissipation of the gate.

The rectifying memristors have large resistance values. Table III shows some of these published values in the literature. The simulation results show that the circuits operate correctly for $T_C$=$T$, where $T_C$ is the clock time of the circuits. For example, Fig. 12a shows the simulation results of applying inputs ($v_a$, $v_b$, $v_c$) = (1, 0, 1) to the adder in Fig. 7. The outputs are $s$ and $c$ stored as resistance in two memristors in crossbar C. Voltages on horizontal nanowires of crossbar B exhibit the outputs of wired-OR gates, utilized for implementing the addition operation in the Product-of-Sums form. To assess the

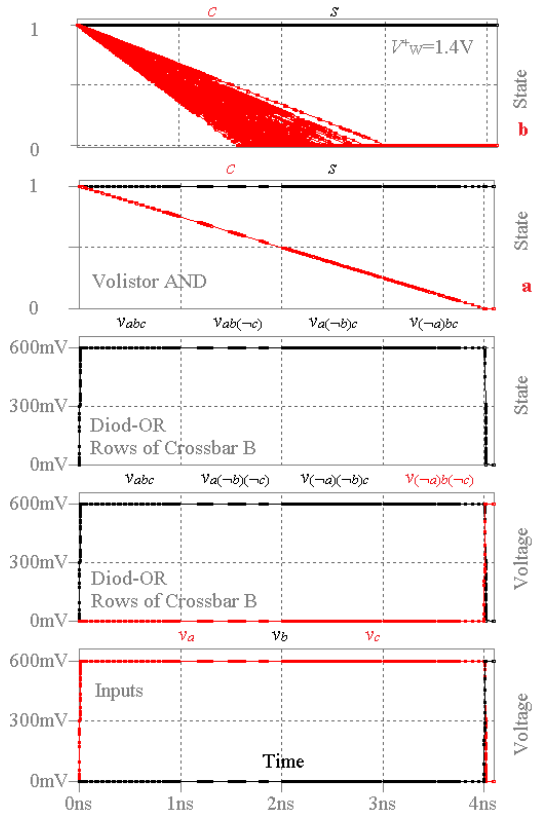


Fig. 12. Simulations of the 1-bit adder shown in Fig. 7. (a) Propagation of inputs through the crossbar arrays and calculation of intermediate signals are shown. $v$ denotes a voltage value corresponding to a literal, product, or sum–of–products. Inputs are $v_a$, $v_b$, and $v_c$, and outputs are $s$ and $c$. (b) Monte Carlo simulations show the circuit operation for varied threshold and programming voltages with means 1V and 1.2V and standard deviation 0.05.

**TABLE III**
**EXAMPLES OF RESISTANCE VALUES OF RECTIFYING MEMRISTORS**

| Ref | $R_{ON}$ | $R_{OFF}$ | Rectifying Ratio |
|---|---|---|---|
| [14] | ≈3V/10$^{-3}$A | ≈3V/10$^{-7}$A | $10^5$* |
| [19] | 100KΩ-1MΩ | 75MΩ-750MΩ | - |
| [20] | ≈3V/10$^{-10}$A | ≈3V/10$^{-12}$A | $10^5$ |
| [29] | ≈0.5V/40×10$^{-9}$A | ≈0.5V/10$^{-9}$ A | >$10^6$ |

*When a bias voltage is larger than1.5V.

tolerance of the circuit to variations in threshold and programming voltages, we ran Monte Carlo simulations 200 times. The varied threshold and programming voltages were sampled from normal distributions with means 1V and 1.2V and standard deviation 0.05. Fig. 12b shows the state variations in the carry bit memristor for bias voltage $V^{+}_{W}$ is 1.4V demonstrating the correct operation of the circuit.

The switching dynamics in memristors are highly nonlinear, i.e., the switching time varies exponentially with the voltage. Thus, voltages close but not higher than the threshold will induce some switching that changes the voltage on a wired logic. The model used, as mentioned above, assumes linear switching dynamics for memristors. This assumption, however, has a minor effect on the operations of programmable diode gates (#1, 5, and 6 in Table I) because they use small input and output voltages (i.e., as small as almost half the threshold voltages that induce negligible state drifts in memristors). The model also has a minor effect on the operations of volistor gates because they use the same voltage configuration at the input as programmable diode gates. As a result, state drifts in memristors connected to inputs are negligible. The switching can only occur in the output memristors, which are always reverse-biased. This switching would further increase the resistances of the output memristors (as opposed to just been reverse-biased) [20] and would further decrease reverse currents, which impact the wired-logic connecting the output memristors to other memristors driven by the inputs (see e.g., volistor NOR in Fig. 5b). The switching dynamics in a rectifying memristor for HRS and LRS are different, and both volistors and programmable diode gates were designed to benefit from this device characteristic.

### B. Size

Each stack in the proposed architecture consists of $m\times n\times l$ memristors where $l$ is the number of crossbar arrays. The number of transistors connected to each nanowire is assumed to be five (as described above), therefore, the number of transistors connected to the crossbar arrays is $(m+n)\times 5l$. In addition, the number of transistors in peripheral switches for connecting stacks of adjacent crossbar arrays is $(m+n)\times 20l$, where the two switches on each side of the stack are non-inverting tristate buffers and transmission gates, and each non-inverting tristate buffer and transmission gate consist of ten transistors. Concluding that the number of transistors connected to each stack is $(m+n)\times 25l$, i.e., five for the voltage levels and 20 for the interconnect tristate buffers and transmission gates. Therefore, the number of transistors to memristors ratio is $25(m+n)\times l$: $m\times n\times l$ or approximately 7.74:1 for the size of crossbar arrays used above. When the size of the crossbar arrays increases, the number of transistors to memristors decreases. For example, for a stack of $m\times n$ crossbar arrays, when $m=n$, the transistor to memristor ratio becomes 50:$n$. When $n>50$, this ratio becomes smaller than one. Fig. 13 shows the number of transistors to memristors ratio for different sizes of crossbar arrays demonstrating the area behavior of the 3D architecture for large crossbar arrays. The area behavior of the 3D architecture comes not only due to the over numbered memristors versus transistors, but also the size of the memristors is smaller than transistors.

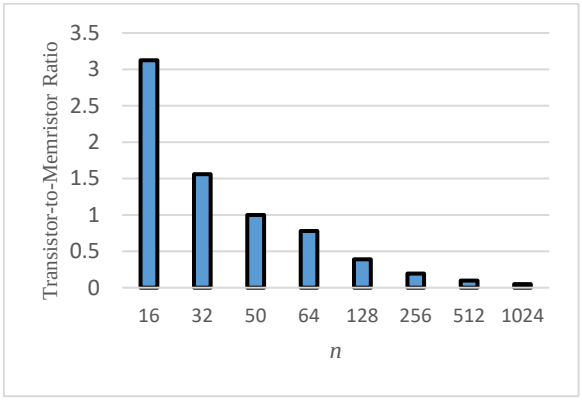


Fig. 13. Transistor- to-memristors ratio for the 3D crossbar arrays of $n\times n\times l$ where $l$ has arbitrary value.

### C. Delay

This paper assumes a clock time of $T_C$, which is sufficient to toggle the state of a memristor under $V_{SET}$ or $V_{RESET}$. In the first example, the 1-bit full adder, the circuit was realized in $3T_C$ including the initialization of output memristors to HRS and then to LRS and the computation. In the second example, the 4-bit adder was implemented in $9T_C$ that includes the initialization and programming of the input memristors ($4T_C$), and the computation of the carry and sum bits ($5T_C$). In the third example, the 8-bit multiplexer, the circuit was implemented in $8T_C$ including the initialization and programming of the input memristors ($4T_C$), and the computation ($4T_C$).

Note that in the current memristor technology, speed is a big shortfall of memristor-based logic as opposed to conventional-CMOS logic [31], i.e., reprogramming a memristor, over creating an inversion layer in a transistor channel, requires a significant amount of energy (and thus time) due to its large bandgap of ions. In addition, the switching time of memristors outweighs the RC delay, calculated for estimating the propagation delay of a logic gate in a crossbar array. Specifically, the transit-time in the 50nm process is 9.489ps and no memristor logic family can compete with this unless implemented as resistor-transistor logic [31].

### D. Power

For all examples, the power consumption in the 3D architecture was evaluated based on memristor parameters used in [28] and shown in Table II. In addition, this paper utilizes the following voltage levels and a reference resistor for implementing logic functions: -0.7$V$, 0V, 0.6, 1.4, and 15MΩ. These voltage levels are generated by dividing down a single supply. Although this approach avoids the need for charge pumps and additional power regulation but largely increases the power consumption in the circuit. However, cascading programmable diode gates and volistor gates for logic computation in crossbar arrays avoid static power consumption. This feature does not exist in CMOS circuits and some memristor logic families.

The computational power for implementing the 1-bit full adder in crossbar arrays A1, B1, and C1 and peripheral switches was calculated. For any combination of inputs, except for 000

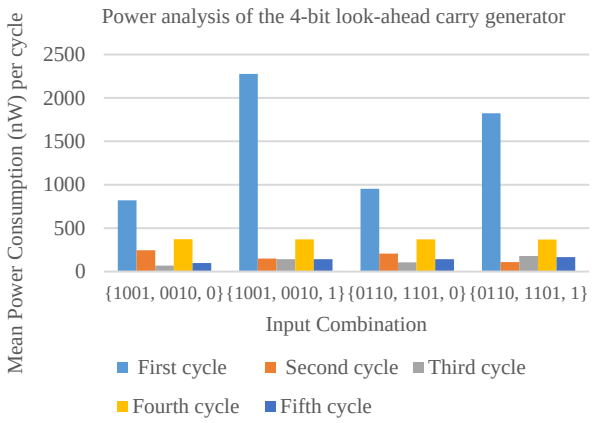


Fig. 14. Power analysis of the 4-bit look-ahead carry generator.

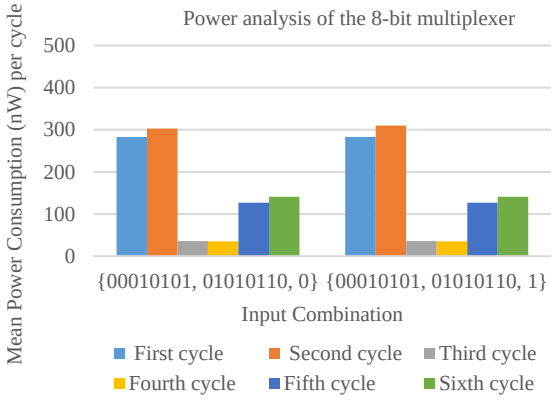


Fig. 15. Power analysis of the 8-bit multiplexer.

and 111, the mean power is about 350 nW. For the other two input combinations, the mean power is about 270 nW. Note that the memristor model used was intended for slow sweep experiments. Typically higher threshold voltages need to be applied for fast switching, which increases the power consumption. For example, the power consumption at $V^{+}_{W}$=1.6V increases by 1.4%.

The computational power for implementing a 4-bit look-ahead carry generator in two stacks of crossbar arrays and the peripheral switches was calculated, as well. Fig. 14 summarizes the power analysis of the circuit for four combinations of inputs *a*, *b*, and $c_0$ over five clock cycles used for implementing the adder (see Section III-C). Note that the adder consumes more power in the first clock cycle as opposed to other clock cycles. In fact, the adder uses more CMOS switches in the first clock cycle than other clock cycles (see Fig. 9a). These switches consume 74-90% of the total power consumed in the first clock cycle. Furthermore, the peak power, which is 2.3 uW, occurs in the first clock cycle.

The computational power for implementing the 8-bit multiplexer in crossbar arrays A1 through E1 and peripheral switches was calculated. Fig. 15 summarizes the power analysis of the circuit for two combinations of inputs *a*, *b*, and $c_0$ over six clock cycles used for implementing the multiplexer (see Section III-D). Note that the power consumed during the initialization step is not shown in Fig. 15. The peak power occurs in the second clock cycle and equals 310 nW. In the previous example, the CMOS switches dominated the power in the first cycle, whereas in this example, the CMOS switches dominate the power in both the first and second cycles.

Table IV shows similarities and differences of multiple 3D architectures. Our circuit trades off some of the area of CMOL in exchange for flexibility (e.g., memory access, programming speed, moving data between crossbars, and programmability of applications).

## V. Conclusion

A combination of a CMOS layer and a 3D stack of memristor crossbar arrays in a 3D architecture provides an innovative approach to utilize the benefit of CMOS, memristors, and 3D circuits. This paper proposes a configurable 3D architecture based on memristor crossbar arrays with a specific focus on connecting the crossbar arrays to the CMOS layer and to each other. The proposed architecture demonstrates programmable flexibility in connecting the stacked crossbar arrays. The proposed architecture also utilizes programmable flexibility for implementing completely different functions such as RAM arrays, logic gates, etc. The proposed connections facilitate changing the architecture topology, which allows data to move (be processed) in planes perpendicular to the stacked crossbar arrays. In addition, these connections make connecting stacks of crossbar arrays are feasible as demonstrated in Example 2 (Fig. 8). Simulations were conducted, and area, delay, and power analysis were shown demonstrating the behavior of the proposed architecture. The goal of our architecture is increased flexibility for programming circuits, functions, and interconnects, not the comparison of size, performance, and power to other architectures. Our architecture is suitable for pipeline and parallel computing and enables in-memory computations as examples show.

## References

[1] K. K. Likharev, and D. B. Strukov, "CMOL: Devices, circuits, and architectures," in *Introducing Molecular Electronics*, pp. 447-477. Springer, Berlin, Heidelberg, 2006.

[2] L. Chua, "Memristor-The missing circuit element," in *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507-519, Sep 1971.

[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams., "The missing memristor found," in *Nature*, vol. 453, no. 7191, pp. 80-83, 2008.

[4] M. D. Pickett, and R. Stanley Williams, "Sub-100 fJ and sub-nanosecond thermally driven threshold switching in niobium oxide crosspoint nanodevices," in *Nanotechnology*, vol. 23, no. 21, pp. 215202, 2012.

[5] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. Stanley Williams, "Sub-nanosecond switching of a tantalum oxide memristor," in *Nanotechnology*, vol. 22, no. 48, pp. 485203, 2011.

[6] M. J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, *et al*., "A fast, high-endurance and scalable non-volatile memory device made from asymmetric $Ta_2O_{5-x}/TaO_{2-x}$ bilayer structures," in *Nature materials*, vol. 10, no. 8, pp. 625-630, 2011.

[7] S. Pi, P. Lin, and Q. Xia, "Cross point arrays of 8 nm× 8 nm memristive devices fabricated with nanoimprint lithography," in *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 31, no. 6, pp. 06FA02, 2013.

[8] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, *et al*., "Memristor−CMOS hybrid integrated circuits for reconfigurable logic," in *Nano letters*, vol. 9, no. 10, pp. 3640-3645, 2009.

[9] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," in *Nature nanotechnology*, vol. 8, no. 1, pp. 13-24, 2013.

TABLE IV COMPARISONS OF 3D CIRCUITS DESIGNS

| Comparisons | | 3D circuits | | | |
|---|---|---|---|---|---|
| | | Proposed 3D Circuit | CMOL [1][12] | FPNA [13] | [14] |
| Sneak current Eliminated by | | Rectifying memristors | Biasing nanowires[34] | Biasing nanowires | Rectifying memristors |
| Connecting Crossbars-to-CMOS | | Extended Nanowires-to-CMOS | Nano-Pins (with different sizes) underneath crossbar | Pads and Pins (much larger than CMOL Nano-Pins) underneath crossbar | Peripheral Contact Pads |
| Applications | | Memristive Logic, memory, etc. | CMOS Logic, memory, etc. | CMOS Logic (routing in crossbar) | Memristive Logic, memory, etc. |
| Memory access | | 1-word per crossbar | 1-bit per stacked crossbars [1] | - | 1-word per crossbar |
| Programming/mapping speed | | 1-word per crossbar per cycle | 1-bit per stacked crossbars per cycle | 1-bit per crossbar per cycle | 1-word per crossbar per cycle |
| Computation speed | | Slow (in crossbars as opposed to CMOS) | Fast | Fast | Slow (in crossbars as opposed to CMOS) |
| Size | #vias | Large ($2nl$ per crossbar) [2] | Small ($2n^2$ per stacked crossbars) | Small ($2n^2$ per stacked crossbar) | Large ($2nl$ per crossbar) |
| | # via translation layer (above CMOS) | 0 | $l$-1 | 0 | 0 |
| | # external selector | Not required | Cannot be used | Cannot be used | Not required |
| | Overhead used for memory application | 5-transistor per each nanowire | 4 decoders per stacked crossbars | - | - |
| | CMOS Overhead for logic implementation | 5-transistor per nanowire [3] | CMOS logic gates and flip flops | CMOS logic gates | - |
| | Overall | Larger | Smaller | Small | - |
| Flexibility | Vias (contact) locations | Flexible | Precise | More flexible than CMOL | Flexible |
| | Data transfer | Extremely flexible (all crossbar planes) | Restricted (one plane) | Restricted (one plane and only for routing) | - |
| | Connecting stacked crossbars | Yes | No | - | Set at fabrication |
| | Programmability of applications | Any array can be programmed for any type of application | No | No | No |
| | Connecting a network of 3D architectures | Feasible | - | - | - |
| Performance (circuit alu4) | Power (uW) | 20.27 | - | 61 | - |
| | Delay (ns) [4] | 36 | 3.3 | 28.7 | - |

**1** Two pairs of decoders are needed to facilitate accessing of one pair of pins independently, which in turn allows accessing one memristor in stacked crossbar arrays.
**2** For $nl$, $n \times n$ is the size of a crossbar array, and $l$ is the number of crossbar arrays in a stack.
**3** Five transistors per nanowire are sufficient for both logic and memory applications.
**4** Calculation delay of alu4 in CMOS FPGA was reported 5.1ns [35].


[10] I. Valov, R. Waser, J. R. Jameson, and M. N. Kozicki, "Electrochemical metallization memories—fundamentals, applications, prospects," in *Nanotechnology*, vol. 22, no. 25, pp. 254003, 2011.

[11] I. A. B. Adames, J. Das, and S. Bhanja, "Survey of emerging technology based physical unclonable functions," In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, IEEE Press. 317-322, 2016.

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

[13] G. S. Snider, and R. Stanley Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," in *Nanotechnology*, vol. 18, no. 3, pp. 035204, 2007.

[14] C. Li, and Q. Xia, "Three-dimensional crossbar arrays of self-rectifying Si/$SiO_2$/Si memristors," In *Handbook of Memristor Networks*, pp. 791-813. Springer, Cham, 2019.

[15] K. Nomura, K. Abe, S. Fujita and A. DeHon, "Novel Design of Three-Dimensional Crossbar for Future Network on Chip based on Post-Silicon Devices," *2006 1st International Conference on Nano-Networks and Workshops*, Lausanne, 2006, pp. 1-5.

[16] M. M. Shulaker, K. Saraswat, H. -. P. Wong, and S. Mitra, "Monolithic three-dimensional integration of carbon nanotube FETs with silicon CMOS," *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, Honolulu, HI, 2014, pp. 1-2.

[17] M. M. Shulaker *et al*., "Monolithic 3D integration of logic and memory: Carbon nanotube FETs, resistive RAM, and silicon FETs," *2014 IEEE International Electron Devices Meeting*, San Francisco, CA, 2014, pp. 27.4.1-27.4.4.

[18] M. J. Aljafar, M. A. Perkowski, and J. M. Acken, "Volistor Logic Gates in Crossbar Arrays of Rectifying Memristors," in *International Journal of Unconventional Computing*, vol. 14, no. 3-4, pp. 319-348, 2019.

[19] K. K. Kim, *et al*., "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," in *Nano letters*, vol. 12, no. 1, pp. 389-395, 2012.

[20] K. M. Kim, *et al*., "Low-power, self-rectifying, and forming-free memristor with an asymmetric programing voltage for a high-density crossbar application," in *Nano letters*, vol. 16, no. 11, pp. 6724-6732, 2016.

[21] D. Chakraborty, S. Raj, and S. K. Jha, "A compact 8-bit adder design using in-memory memristive computing: Towards solving the Feynman Grand Prize challenge," *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Newport, RI, 2017, pp. 67-72.

[22] H. A. D. Nguyen, J. Yu, L. Xie, M. Taouil, S. Hamdioui, and D. Fey, "Memristive devices for computing: Beyond CMOS and beyond von

Neumann," *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Abu Dhabi, 2017, pp. 1-10.
[23] A. Siemon, S. Menzel, A. Chattopadhyay, R. Waser, and E. Linn, "In-memory adder functionality in 1S1R arrays," *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, 2015, pp. 1338-1341.
[24] M. J. Aljafar, M. A. Perkowski, J. M. Acken and R. Tan, "A Time-Efficient CMOS-Memristive Programmable Circuit Realizing Logic Functions in Generalized AND–XOR Structures," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 23-36, Jan. 2018.
[25] N. G. Murali, P. S. Vardhan, F. Lalchhandama, K. Datta, and I. Sengupta, "Mapping of Boolean Logic Functions onto 3D Memristor Crossbar," *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, Delhi, NCR, India, 2019, pp. 500-501.
[26] B. Chakrabarti, *et al*., "A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit," *Scientific reports*, vol. 7, p. 42429, 2017.
[27] M. Aljafar, P. Long, and M. Perkowski, "Memristor-based volistor gates compute logic with low power consumption," *BioNanoScience*, vol. 6, no. 3, pp. 214-234, 2016.
[28] E. Lehtonen, J. Tissari, J. Poikonen, M. Laiho, and L. Koskinen, "A cellular computing architecture for parallel memristive stateful logic," in *Microelectronics Journal*, vol. 45, no. 11, pp. 1438-1449, 2014.
[29] K-H Kim, S. H. Jo, S Gaba, W Lei, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," in Applied Physics Letters vo. 96, p. 053106, 2010.
[30] H. A. D. Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui and K. Bertels, "On the Implementation of Computation-in-Memory Parallel Adder," IEEE Trans. VLSI, vol. 25, pp. 2206 - 2219, 2017.
[31] X. -Y. Wang *et al*., "High-Density Memristor-CMOS Ternary Logic Family," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 264-274, Jan. 2021.
[32] J. K. Eshraghian, S. M. Kang, S. Baek, G. Orchard, H. H. C. Iu, and W. Lei, (2019, March). Analog weights in ReRAM DNN accelerators. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (pp. 267-271). IEEE.
[33] F. Cai, *et al*. "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, no. 7, pp. 290-299, 2019.
[34] D. B. Strukov and K. K. Likharev, "Defect-tolerant architectures for nanoelectronic crossbar memories," in *Nanosci Nanotechnol*, vol. 7, pp. 151–167, 2007.
[35] D. B. Strukov and K. K. Likharev, "CMOL FPGA circuits," in *CDES*, pp. 213–219, 2006.

**Muayad J. Aljafar** He received his M.Sc. and Ph.D. degrees in electrical and computer engineering from Portland State University in 2016 and 2018. He was a professor at Basra College of Science and Technology since 2019. Recently, he joined Tallinn Technology University (TalTech) as a researcher. His primary research areas are memristor computations and hardware security.

**John M. Acken** He received his BS(76) and MS(78) in electrical engineering from Oklahoma State University and his PhD (88) in electrical engineering from Stanford University. He is a research faculty member in the Electrical and Computer Engineering Department, Portland State University, Portland, OR. Prior to PSU, he taught and guided research at Oklahoma State University. Dr. Acken has contributed to several technical activities including: Session Chair: CCCT 2008, Publicity Chair: IEEE IDDQ Testing Workshop 1996, Session Chair: ACM/SIGDA Workshop on Logic Level Modeling for ASICs 1995.