Overhead Requirements for Stateful Memristor Logic

Xuan Hu, *Student Member, IEEE*, Michael J. Schultis, Matthew Kramer, Archit Bagla, Akshay Shetty, and Joseph S. Friedman, *Member, IEEE*

***Abstract*— Memristors are being explored as a potential technology to replace CMOS for logic-in-memory systems that exploit the memristive non-volatility. Memristors are two-terminal, non-volatile device that exhibit a variable resistance that is dependent on the applied voltage history of the device, providing the capability to store and process information within the same structure. The ability of memristors to perform logic has been previously demonstrated, but previous analyses of memristor logic efficiency have not included the overhead CMOS circuitry that is required to control memristor logic operations. In this paper, the required overhead CMOS circuitry for implementing logic with memristors is evaluated for standard logic gates and a one-bit full adder to enable an analysis of the overall system efficiency. The results show that the number of CMOS devices in the overhead circuitry can be upwards of 50 times that of a conventional CMOS implementation, and that the power-delay product of the memristor logic with overhead circuitry is roughly one billion times greater than for conventional CMOS circuits. These results enable the conclusion that the overhead circuit requirements for stateful memristor logic threaten to negate any efficiency improvements that are achieved by the memristors themselves.**

***Index Terms*— Memristors, memristive systems, logic gates, non-volatile logic computing, control circuit, beyond CMOS computing, memristor control circuit.**

I. Introduction

SINCE the invention of the first integrated circuit, the density of transistors has grown exponentially. While this trend persists, thanks to continued advancements in silicon process technologies, many have posited that the end of Moore's Law is quickly approaching [1]. This eventual barrier to further increases in transistor density encourages investigation into novel circuit architectures. Many concepts for beyond-CMOS computing have been proposed [2]–[15], with significant recent interest toward performing logic-in-memory [16], [17] with memristors [18], [19].

Based on research from HP Labs, a simplified understanding of memristor [20] behavior is depicted in Fig. 1(a), where the shifting of the connection points is influenced by the applied voltage [21]. As a varying voltage is applied between terminals 1 and 2, the resistance of the device shifts between low and high resistance states. Fig. 1(b) shows the symbol that is commonly used for memristor devices. Consistent with the memristor model used in this work [22], a negative voltage applied from the "+" terminal to the "−" terminal decreases the resistance to R_{ON} (*i.e.*. A positive voltage applied from the "+" terminal to the "−" terminal increases the resistance to R_{OFF} (*i.e.*, an open circuit). Memristors based on this behavior have been proven to be able to perform logic both theoretically [18] and experimentally [19], [21].

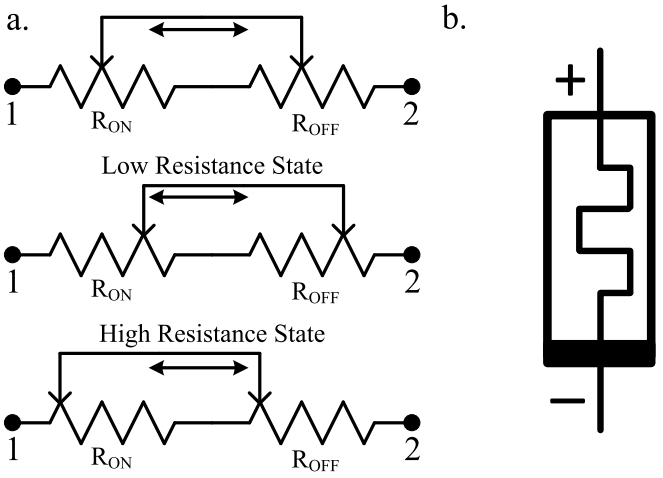


Fig. 1. (a) Simplified memristor behavior (adapted from [21]). (b) Memristor symbol.

Since the publication of the work performed by HP Labs, there has been a flurry of research in this field. This includes the ability of two memristors to perform the implication (IMPLY) function [19], standard functions such as AND and OR [23], and more advanced functions such as the one-bit full adder [24] and multi-bit full adder [18]. Various schemes for memristor connectivity have also been suggested including the parallel [19], [25] memristor and crossbar configurations [26]–[28]. Furthermore, some drawbacks have been identified, such as the potential for sneak-paths in crossbar arrays [18], [29] and the need for sense amplifiers to read the state of a memristor [18].

Amidst all of the excitement toward memristors for non-volatile logic, very serious concerns have been overlooked and minimized. In particular, the peripheral circuitry required

Manuscript received January 16, 2018; revised May 23, 2018 and July 9, 2018; accepted July 20, 2018. Date of publication August 27, 2018; date of current version December 6, 2018. This paper was recommended by Associate Editor W. Zhao. *(Xuan Hu and Michael J. Schultis contributed equally to this work.) (Corresponding author: Joseph S. Friedman.)*

The authors are with the Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: joseph.friedman@utdallas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2018.2861463

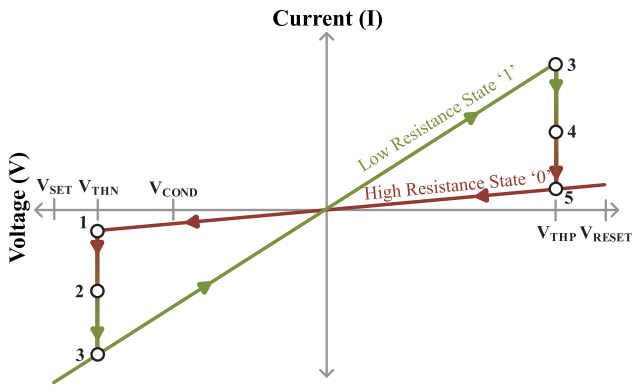


Fig. 2. Ideal thresholded memristor zero-crossing hysteresis curve.

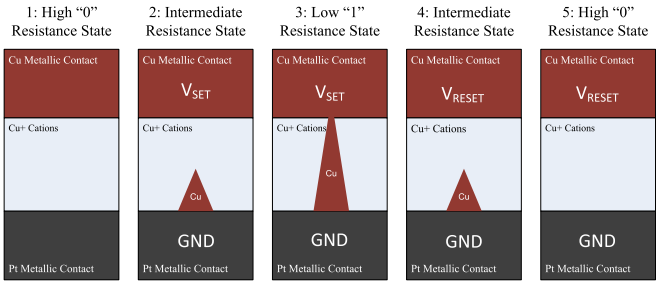


Fig. 3. Bipolar filametary switching process (adapted from [36], [37]).

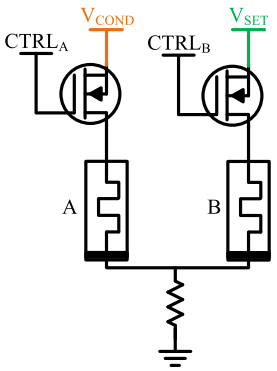


Fig. 4. Stateful memristor logic operation for implication function $A \rightarrow B$.

to control memristor logic has not been fully designed and analyzed [30]. The required CMOS control circuitry poses a serious threat that must be considered in order to enable realistic predictions of the efficiency of this technology.

To truly understand the efficacy of a memristor-based computing system, it is necessary to determine the overhead required to control the application of voltages to the memristors. This paper therefore evaluates the CMOS circuitry required as overhead to control memristor-based functions, providing insight into the true viability of stateful memristor logic. A complete overhead circuit is proposed and analyzed, leading to the conclusion that even if the memristors themselves are ignored, more CMOS transistors are required for the memristor control circuit than are required for a conventional CMOS implementation. Furthermore, the power delay product of a conventional CMOS implementation is greater than one billion times better than with memristors, proving that even with significant improvements in memristor-based architectures, CMOS-based computing is more efficient than memristor logic.

II. Overview of Memristor Logic

To provide insight into the operation of memristors from a physical perspective, memristor technology is introduced. The IMPLY function is explained as the basis for computing with memristors. Additional Boolean functions are considered for implementation with memristors, and it is shown that if the control circuit is ignored, fewer memristors are required as compared to the number of transistors in a conventional CMOS implementation. (In Section III, it is shown that when the control circuit is not ignored, memristor logic requires many additional devices).

A. Memristor Technology

While there are many types of memristors, the most common has a zero-crossing curve with hysteresis [21], [31]–[33]. In the ideal I-V curve shown in Fig. 2, the device changes to a low resistance state when the voltage applied across the device (V_{SET}) is less than the V_{THN} voltage. Conversely, the device reverts to the high resistance state when the voltage applied across the device (V_{RESET}) is greater than the V_{THP} voltage. The conditional voltage, V_{COND}, is also critical to logic operation using memristors and is described in the next section.

The physical realization of memristors varies based on the specific construction of the memristor, and different mechanisms have been discussed previously [34], [35]. As an introduction, the discussion here is limited to the bipolar filamentary switching process. A simplified diagram is depicted in Fig. 3 to illustrate the mechanism of resistance change in the device [36], [37]. When the V_{SET} voltage is applied to the device, a metallic filament is formed, switching the device to the low resistance state (logic "1"). When the V_{RESET} voltage is applied to the device, the filament retreats, switching the device back into the high resistance state (logic "0"). This property can be exploited to perform logical operations such as the implication function, as depicted in Fig. 4 and described further in Section II.B.

As the device switching is "mechanical" in the sense that ions are required to move to form a filament, the switching speed is limited by the velocity of the ions. Though the switching speed may be limited, the mechanical switching enables these devices to retain their state without external excitation. This quality makes memristors a promising technology, as they can both perform logical functions as well as act as local memory storage, reducing long-distance data transfer and overcoming the von Neumann bottleneck [34]. This non-volatile memristor device can be used both to store information as well as perform complex logic functions [18], [19], [23], [24], with the implication function as the basis for memristor logic.

TABLE I
IMPLY FUNCTION TRUTH TABLE AND NODE VOLTAGES

	Input		During Operation		Output
Case	A	B	V_{PD}	V_B	B' = A → B
1	0	0	GND	V_{SET}	1
2	0	1	V_{SET}	0	1
3	1	0	V_{COND}	V_{SET}-V_{COND}	0
4	1	1	$V_{SET} < V_{PD} < V_{COND}$	0	1

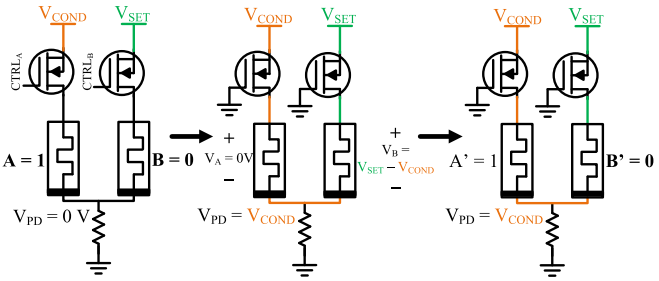


Fig. 5. Implication logic operation for case 3 of Table I.

B. Implication Function

The IMPLY gate is the fundamental building block for stateful memristor logic. The IMPLY function is commonly denoted by → or IMP (*e.g.*, $A \rightarrow B$ or A IMP B), and is also known as the material conditional [19]. The A IMP B function can also be understood as standard Boolean logic function NOT(A) OR B (*i.e.* $\overline{A} + B$). This function is not commutative, and the four input cases are shown in the truth table of Table I. The inputs of the IMPLY function are the states of the memristors A and B, and the output is the state of B after the operation (B'). The state of memristor A is unchanged during the IMPLY operation.

Fig. 5 depicts the functional operation of the IMPLY gate for case 3, illustrating the importance of the V_{COND} voltage. As described in Fig. 2, the memristor switches states from high resistance (0) to low resistance (1) when the V_{SET} voltage is applied across its terminals from + to −. The V_{COND} voltage is of less magnitude than V_{SET}, such that applying V_{COND} or (V_{SET} - V_{COND}) does not change the memristor from high resistance (0) to low resistance (1). Table I summarizes each of the four combinations of inputs for the IMPLY gate, noting the voltages applied and the states of each of the memristors before, during, and after the operation.

C. Stateful Memristor Logic

The IMPLY gate can be extended to perform additional logic functions with the addition of parallel memristors. It has been shown in [19] that a NAND operation can be performed, and in [18] that the AND, OR, NOR, and XOR gates can be implemented with serial applications of the implication function. Beyond the advantage of non-volatility in memristors, an additional benefit to memristors is that fewer devices are required to perform standard logic functions (when the control overhead is not included). Table II summarizes the number of transistors required compared to the number of memristors required for equivalent functions.

TABLE II
FUNCTION DEVICE COUNT WITHOUT CONTROL CIRCUITRY

Function	CMOS Only	Memristor
IMPLY	6	2
AND	6	4
OR	6	4
NAND	4	3
NOR	4	3
XOR	8	5
XNOR	8	4
Full Adder	28	6

Table II shows that memristors have a clear advantage over standard CMOS when considering the number of devices required to perform the logical function itself. When using memristors, the standard Boolean functions may require only one-third as many the number of devices required for CMOS, and the full adder circuit requires 22 fewer devices, a nearly 80% reduction. However, this comparison only tells part of the story.

A significant drawback to the six-memristor full adder implementation is that the function requires 29 serial steps, signficantly slowing down the operation time [18]. In addition to the significant reduction in operating speed due to the mechanical switching of memristors and the number of serial steps, this device comparison does not fully consider all circuitry required. While it has been shown that these Boolean functions can be performed using fewer memristors (as compared to transistors) when the overhead circuit is ignored, little research has gone into fully understanding the required supporting CMOS circuitry necessary to coordinate sequential operations in memristor-based logic arrays.

D. Alternative Memristor Logic Styles

This paper focuses specifically on the IMPLY-based logic of [18], [19], and [25]. Many other memristor logic styles have been proposed, including:

- Memristor ratioed logic (MRL) [38],
- Memristor-aided logic (MAGIC) [39],
- Complementary resistive switch (CRS) logic [40], [41],
- Crossbar array logic [42], and
- Memristors-as-drivers (MAD) logic [43], [44].

The voltages applied to the memristors must be carefully coordinated by non-memristor devices presumably CMOS transistors in all of these configurations. Therefore, the challenges identified in this paper may extend to other memristor logic styles.

III. OVERHEAD CIRCUITRY FOR BOOLEAN FUNCTIONS

As noted above, the design of peripheral circuitry for memristor logic has not been previously proposed [30]. To evaluate

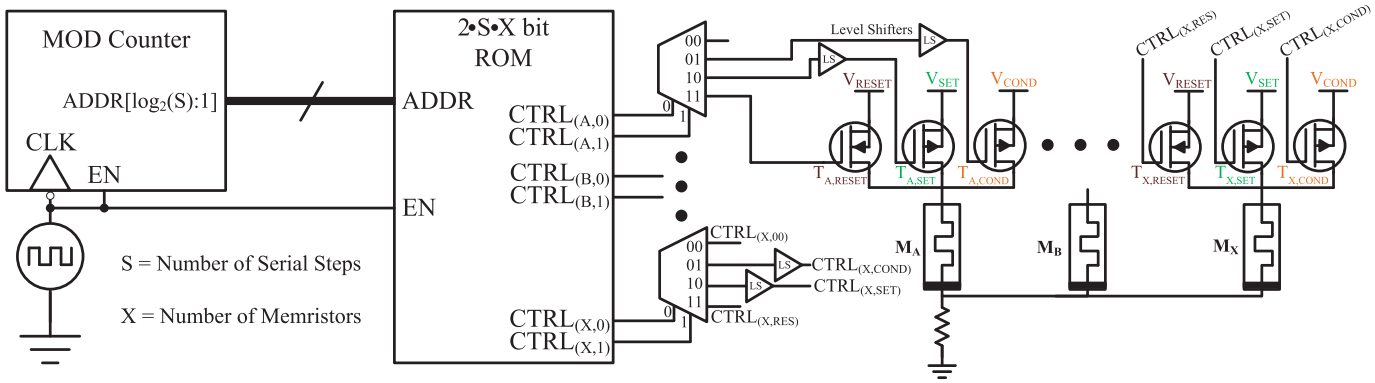


Fig. 6. Block diagram for memory-based, clocked control scheme that scales with memristor requirements. The V_{SET}, V_{RESET}, and V_{COND} voltages must be able to be applied to each memristor to make the array fully programmable, so CMOS transistors are utilized to enable the application of these voltages. A ROM containing the serial step information to implement the desired function is used to drive a set of decoders, which control the CMOS transistors. A system clock increments a counter that causes the ROM to step through a sequence of instructions, applying the correct signals to implement any function with the set of parallel memristors. This block diagram is a fully scalable control block that can be used to control any number of memristors with any number of steps for the computation.

the efficacy of a memristor-based logic system, the CMOS circuitry required to implement memristor logic functions is described. Simple functions such as the IMPLY, AND, OR, and XOR are analyzed to investigate the number of transistors required for memristor-based gates, and the result is compared to the number of transistors required for a purely CMOS gate implementation.

As there is a trade-off between flexibility and control circuit overhead, this paper examines IMPLY-based structures due to their relatively simple overhead control circuits resulting from their relative logical rigidity as compared to other memristor logic proposals [24]. The overhead circuits for the more flexible memristor logic structures are even more complex and cause further negative impact on system efficiency.

A. Overhead Control Circuit Structure

As described in relation to Fig. 5, the memristor logic operations are not performed solely with memristors. Additional transistors are required to apply the appropriate voltages to the memristors to perform the IMPLY function. The structure of Fig. 5, however, is a simplification of the complete structure required for computing with memristors. This particular circuit can perform the IMPLY function only once, and the memristors cannot be reset as a V_{RESET} voltage cannot be applied. Additional transistors are therefore required for each memristor to allow for resetting, making functions repeatable. Furthermore, a set of clock pulses is required to perform sequential steps correctly. Taking these factors into consideration, a memory-based, clocked control scheme that scales with function complexity is demonstrated here. This control scheme is designed to ensure that any sequential memristor-based function can be implemented with this technique.

The framework for a generic memristor array control method has been highlighted in previous work, as depicted in Fig. 3(a) of [30]. This work utilizes an on-chip controller to drive decoders that control the memristor array; however, this framework does not include analysis of the number of supporting transistors required to implement the framework. The control method of [45] incorporates the enhancements highlighted in [30] to be suitable for controlling memristor logic. These enhancements include the ability to select rows and columns to enable parallel execution and the ability to select additional voltages beyond the V_{SET}, V_{COND}, and V_{RESET} voltage nodes previously discussed. The peripheral circuit analyzed here does not allow for parallel execution and also has a limited subset of voltages, so therefore is a simpler version of what is considered necessary for complete memristor-based logic control. Given the simpler structure of the architecture highlighted below, the actual number of overhead transistors required to control a more flexible memristor-based logic array is significantly higher.

The control circuit is centered on read-only memory (ROM) that contains the information required to perform a specific function, as depicted in Fig. 6. A clock increments a MOD-N counter, which drives the input address of the ROM. A MOD-N counter is used in this design to help reduce the number of clocks required to address the ROM, so that only a single clock is required instead of a complex system of clocks. While the system could alternatively be based solely on a multi-phase clock, this requires a significantly more complex clock generation scheme. The ROM interprets the N-bit address and outputs the data contained at this address to drive the inputs of a decoder. The outputs of the decoder are connected to the gates of the transistors (or power switches) that connect the appropriate voltage to the corresponding memristor. The decoder aids in reducing the size of the ROM and also provides the drive strength necessary to quickly drive the gates of the power switches. Additionally, a pair of level shifters are required to drive the V_{SET} and V_{COND} transistors. This complete system coordinates a set of memristors to perform a desired function based on a single clock and a repeatable instruction set.

B. Overhead Circuit Transistor Count

For the computation of any stateful memristor logic function, the necessary control circuit requires the use of a larger

number of transistors than is required for a conventional CMOS implementation. The conventional CMOS NAND and NOR functions require only four transistors; the stateful memristor logic control circuit requires a larger number of transistors for the control circuit consisting of a counter, read-only memory (ROM), a decoder, level shifters, and drivers. This is mathematically proven below; optimizations are made for stateful memristor logic systems with a large number of steps, as it is readily apparent that stateful memristor logic systems with a small number of steps require more transistors than conventional CMOS.

The static counter used in this work utilizes 28 transistors per counter stage. As the number of stages is equivalent to $\log_2(S)$ for a stateful memristor logic function with S steps, the number of transistors required by the counter is

$$N_{\text{COUNT}} = 28\log_2(S). \tag{1}$$

In a system with a large number of sequential steps, the transistor cost of the counter is less significant than in a small system.

The sequential operations inherent to stateful memristor logic requires ROMs to store the operation codes that set the memristor driving circuits to the desired modes. For each memristor at each time step, two bits of information are required to encode the voltage applied to the memristor among the four possibilities: floating, conditional, set, and reset. Therefore, a two-bit ROM is required for each memristor at each time step. The minimum core ROM size for any particular function is therefore

$$N_{\text{ROM,CORE}} = 2XS, \tag{2}$$

where X is the number of memristors required for that function. Additionally, an X-row ROM utilizes a $\log_2(X)$-bit address signal, which requires a $\log_2(X)$-to-X decoder for the ROM word lines. Furthermore, an S-column ROM requires S pre-charge PMOS transistors to pull-up the bit line and a sense amplifier to read-out the bit stored in the ROM. A five-transistor differential amplifier is the simplest sense amplifier structure, though most practical sense amplifiers consist of a larger number of transistors. For a ROM with X rows and S columns, the peripheral circuits therefore require at least

$$N_{\text{ROM,PERIPHERAL}} = 6X + 6S - 2 \tag{3}$$

transistors. The total number of transistors required by the ROM is therefore at least

$$\begin{aligned} N_{\text{ROM}} &= N_{\text{ROM,CORE}} + N_{\text{ROM,PERIPHERAL}} \\ &= 2XS + 6X + 6S - 2. \end{aligned} \tag{4}$$

While the other components of the control circuit add significantly to the overhead cost, the transistor count of this ROM alone is greater than a conventional CMOS logic circuit and is a fundamental cause of the inefficiency of stateful memristor logic.

In order to minimize the size of the ROM of a function that requires many steps (as in any large-scale system), the two-bit encoding described above is used to represent the four voltage levels applied to each memristor rather than four bits as would be required by a naïve design. It is therefore necessary to use 2:4 decoders for each memristor, at a cost of 22 transistors each, for a total of

$$N_{\text{DEC}} = 22X. \tag{5}$$

If instead the ROM signals are directly used as the inputs to driving circuits without encoding, the required ROM size is doubled to

$$N_{\text{ROM,NoDecoder}} = 4XS + 12X + 6S - 2. \tag{6}$$

Therefore, decoders are used in this analysis.

As negative voltages are used for the implication operation, voltage level shifters are required to deliver both the V_{Cond} and V_{Set} voltages to each memristor. Each level shifter requires ten transistors, for a total transistor count of

$$N_{\text{LS}} = 2(10X) = 20X. \tag{7}$$

Finally, driver transistors are required to connect each of the three possible memristor voltages (V_{Cond}, V_{Set}, and V_{Reset}) to the supply rails. While these transistors must be relatively large, only one transistor is required for each of these voltages for each memristor, for a transistor count of

$$N_{\text{DRIVE}} = 3X. \tag{8}$$

The total transistor count of the stateful memristor logic control circuit is given by

$$\begin{aligned} N_{\text{TOTAL}} &= N_{\text{COUNT}} + N_{\text{ROM}} + N_{\text{DEC}} + N_{\text{LS}} + N_{\text{DRIVE}} \\ &= 28\log_2(S) + 2XS + 51X + 6S - 2. \end{aligned} \tag{9}$$

Therefore, for additional sequential steps with a given number of memristors, each memristor requires the following number of transistors:

$$N_{\text{STEP}} = 28\log_2(S) + 2XS + 6S. \tag{10}$$

Therefore, as an implication operation requires one step with two memristors, the cost of one implication operation is at least ten CMOS transistors. (This assumes that the counter size is not increased and that all necessary memristors and the corresponding CMOS control circuitry are already included in the system and can be directly used for computation.) As the simplest two-input logic functions (NOR and NAND) require four transistors in a conventional CMOS implementation, the number of transistors in the stateful memristor logic control circuit is always larger than a conventional CMOS logic circuit.

C. CMOS Requirements for Boolean Functions

Only two memristors and a single step are required to perform the IMPLY operation. However, the ROM requires 14 transistors, the decoder requires 44 transistors, the level shifters require 40 transistors, and there are six control transistors, totaling 104 supporting transistors which is nearly 20 times the amount required to perform the same function in traditional CMOS. This result includes ROM optimizations and scales up with the number of memristors and instructions required, making this additional overhead burdensome.

TABLE III
NUMBER OF DEVICES REQUIRED TO PERFORM STANDARD FUNCTIONS WITH SUPPORTING TRANSISTOR COUNT FOR MEMRISTORS

Function	CMOS Only	Memristor + CMOS
IMPLY	6	$2 + 104$
AND	6	$4 + 292$
OR	6	$4 + 285$
NAND	4	$3 + 231$
NOR	4	$3 + 196$
XOR	8	$5 + 473$
XNOR	8	$4 + 410$
Full Adder	28	$6 + 676$

As shown in Table III, the IMPLY function requires fewer devices as compared to standard CMOS; however, when the supporting CMOS structure for memristor logic is considered, the IMPLY gate requires many additional transistors.

With the structure in Fig. 6, it is clear that a large number of additional CMOS transistors are required for operating a scalable memristor-based computation. The same control method is used to evaluate the number of transistors required for other standard Boolean functions including NAND, AND, NOR, OR, XOR, and XNOR. Each structure scales up with the number of steps required in addition to the number of memristors. Table III summarizes the number of memristors and additional transistors required for the memristor architecture relative to the number of transistors required for a pure CMOS implementation.

A simple NAND gate that typically only requires 4 transistors in a pure CMOS implementation requires 231 additional transistors when computing the function utilizing three memristors. It is clear from the table that a large supporting structure is required for computing with memristors, which can be a major barrier to using memristors to perform logic; even with significant potential improvements, the overhead requirements remain overwhelming.

IV. NAND AND FULL ADDER OVERHEAD ANALYSIS

To demonstrate the challenges resulting from the overhead circuitry, memristor NAND and full adder circuits are thoroughly analyzed. The simulation modeling methods, circuit design, and simulation results are provided, and the overhead circuitry is compared to a standard CMOS implementation. Additionally, the power-delay product, a metric of the efficiency of a logic system, is analyzed for memristor-based logic and compared to a CMOS implementation.

A. Simulation Approach

In this work, the VTEAM memristor model [22] is used to simulate the memristors in the circuit. The IBM 130 nm CMOS technology is integrated with the VTEAM model to design and simulate the CMOS overhead circuit, as well as for the conventional CMOS implementation. The ROM is simulated in a behavioral manner and does not impact the electrical simulation results; the overhead circuit performance and efficiency should therefore be considered optimistic. The VTEAM memristor model, described in Table IV [22], provides a threshold of -0.85 V for switching from high resistance to low resistance; in other words, a negative voltage applied to the positive terminal of the device reduces the resistance. The VTEAM memristor model has a threshold of 0.22 V for switching from low resistance back to high resistance; that is, a positive voltage greater than 0.22 V increases the resistance. While the model utilized here is not a representative example of the fastest memristors demonstrated to date [46], the gap in efficiency shown in section IV.D demonstrates that these fabrication improvements are inconsequential.

TABLE IV
MEMRISTOR NAND FUNCTIONAL SEQUENCE

Step	Operation	Outcome
0	Reset(M_Y)	$M_Y = 0$
1	$M_A \rightarrow M_Y$	$M_Y' = M_A \rightarrow M_Y$
2	$M_B \rightarrow M_Y'$	$M_Y'' = M_B \rightarrow (M_A \rightarrow M_Y) = M_A$ NAND M_B

TABLE V
VTEAM MODEL PARAMETERS

Parameter	Description	Value
R_{OFF}	Off State Resistance	10 kΩ
R_{ON}	On State Resistance	50 Ω
V_{ON}	On Threshold Voltage	-0.850 V
V_{OFF}	Off Threshold Voltage	0.220 V
D	Maximum Filament Height	3 nm
k_{ON}	Model Constant (Filament On)	0.01
k_{OFF}	Model Constant (Filament Off)	5
α_{ON}	Model Constant (Filament On)	2
α_{OFF}	Model Constant (Filament Off)	1

B. Memristor NAND Operation With Overhead Circuit

Similar to CMOS, the NAND gate implementation is the simplest standard Boolean logic function that can be performed with memristors. The NAND gate implementation here is adapted from [19] and requires only three memristors and two steps. As shown in Table V, the IMPLY function is peformed between each of the input devices and the output device sequentially. As noted previously in Table I, if the input memristor is in the high resistance state (0), the output memristor will switch from high resistance (0) to low resistance (1). For the NAND function, if either input memristors is in the high resistance state (0), then the output memristor will switch to the low resistance state (1). On the other hand, if both inputs are in the low resistance state (1), then the output will remain in the high resistance state (0), thereby performing the NAND function.

The result of the $A = B = 0$ input combination for the NAND gate simulation is depicted in Fig. 7, where both inputs start in the high resistance state (0). The power switch control

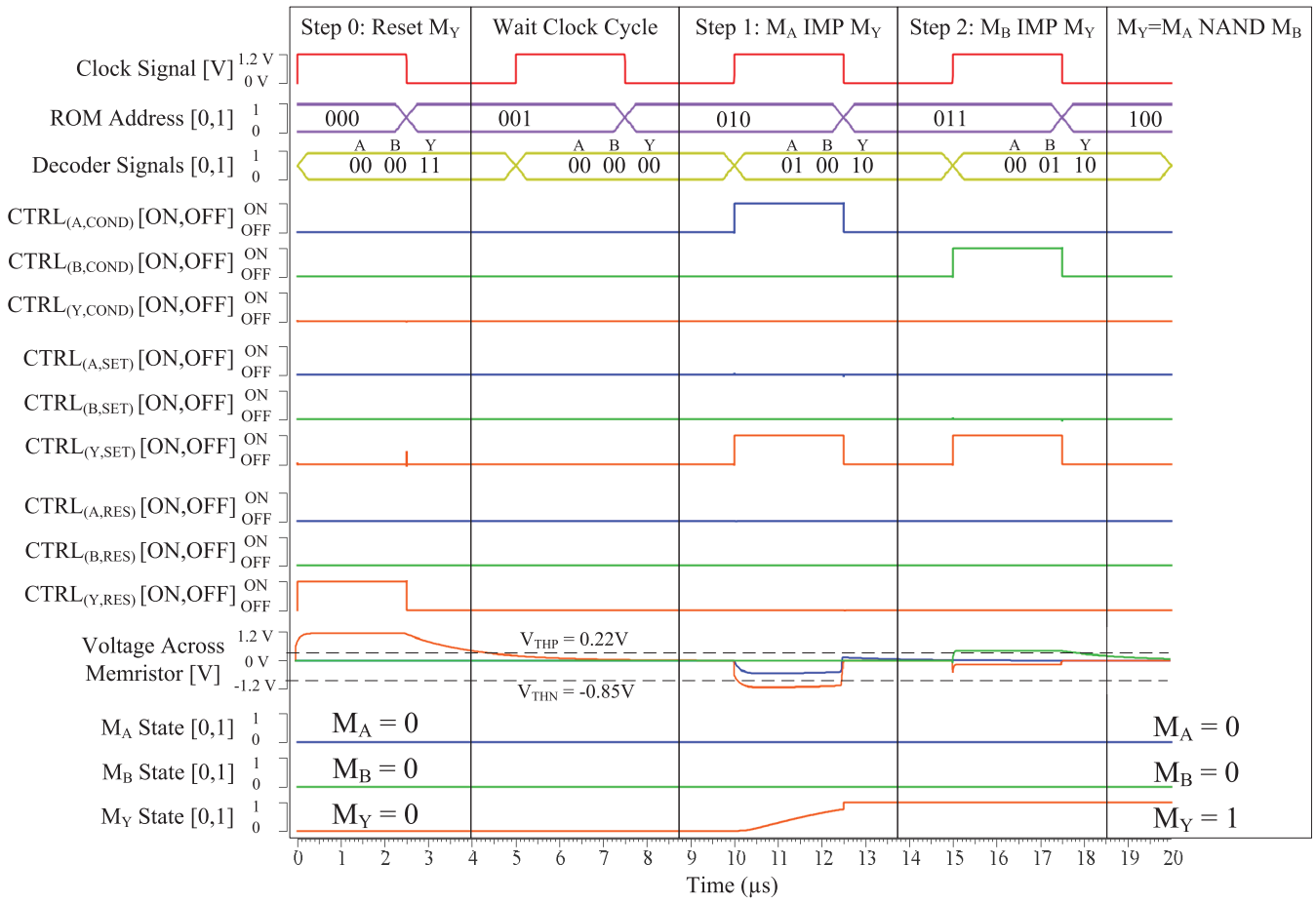


Fig. 7. Simulation of memristor NAND gate. Three memristors M_A, M_B, and M_Y perform the function M_A NAND M_B where memristor M_Y contains the output of the operation. The clock pulse triggers an update of the ROM address on the falling edge and enables the ROM and decoder when high. The ROM address and the input signals to each of the decoders are displayed, in accordance with the structure of Fig. 6. Each memristor has three control transistors for applying V_{COND}, V_{SET}, and V_{RESET}. The voltage of the control signals for each of the nine transistors and the resulting states of the three memristors is shown, where a high signal in the waveform corresponds to turning on the switch that applies V_{COND}, V_{SET}, or V_{RESET} to the A, B, or Y memristor. The analog voltage across each of the memristors illustrates how each of the memristors changes states based on the V_{THP} and V_{THN} threshold voltages. At the end of the simulation, the state of each of the memristors exhibits the correct calculation of the function.

signals are shown for all nine transistors to illustrate how each of the voltages are applied to the corresponding memristor. The first step depicts how the output device, M_Y, is set to 0 by turning on the $T_{Y,RESET}$ transistor. The second step is a buffer step to ensure the output device is reset properly due to the slow switching speed of the memristor. Steps three and four show how the $T_{A,COND}$ and $T_{B,COND}$ transistors are enabled for the input devices, M_A and M_B, and the $T_{Y,SET}$ transistor is enabled for the output device, M_Y, to perform the implication function as described previously in Fig. 5 and Table II. It can be seen from Fig. 7 that during the first IMPLY step, $M_A \rightarrow M_Y$, the output switches from the high resistance state (0) to the low resistance state (1). The second IMPLY step, $M_B \rightarrow M_Y$ also can change the output from the high resistance state (0) to the low resistance state (1), however, in this example the state already changed during the first step, so the output remains in the low resistance state (1). The correct result of the NAND gate is shown at the end of the simulation, with the output in the low resistance state (1).

As mentioned previously, the memristor undergoes a state change through the motion of ions, which can be a slow process. Note the long delay in Fig. 7 for the memristor to switch states, in addition to the slow speed of the entire function. The device requires between 2 and 3 μs to switch, and the entire function requires nearly 20 μs to perform, as compared to a standard CMOS implementation that is achieved within nanoseconds. Even if the model utilized a significantly faster memristor, it would still be much slower than a CMOS transistor.

C. Memristor One-Bit Full Adder With Overhead Circuit

A more thorough analysis of the overhead circuit requirements is achieved through the design and simulation of a memristor one-bit full adder and the corresponding overhead. The memristor one-bit full-adder function is adapted from Kvatinksy *et al.* [18], which uses a six memristor structure to perform the function in 29 serial steps. The memristor circuit is depicted in Fig. 9, utilizing the overhead structure described previously in Fig. 6.

The simulation results are depicted in Fig. 8, showing the evolution of the memristor states through the 29 steps.

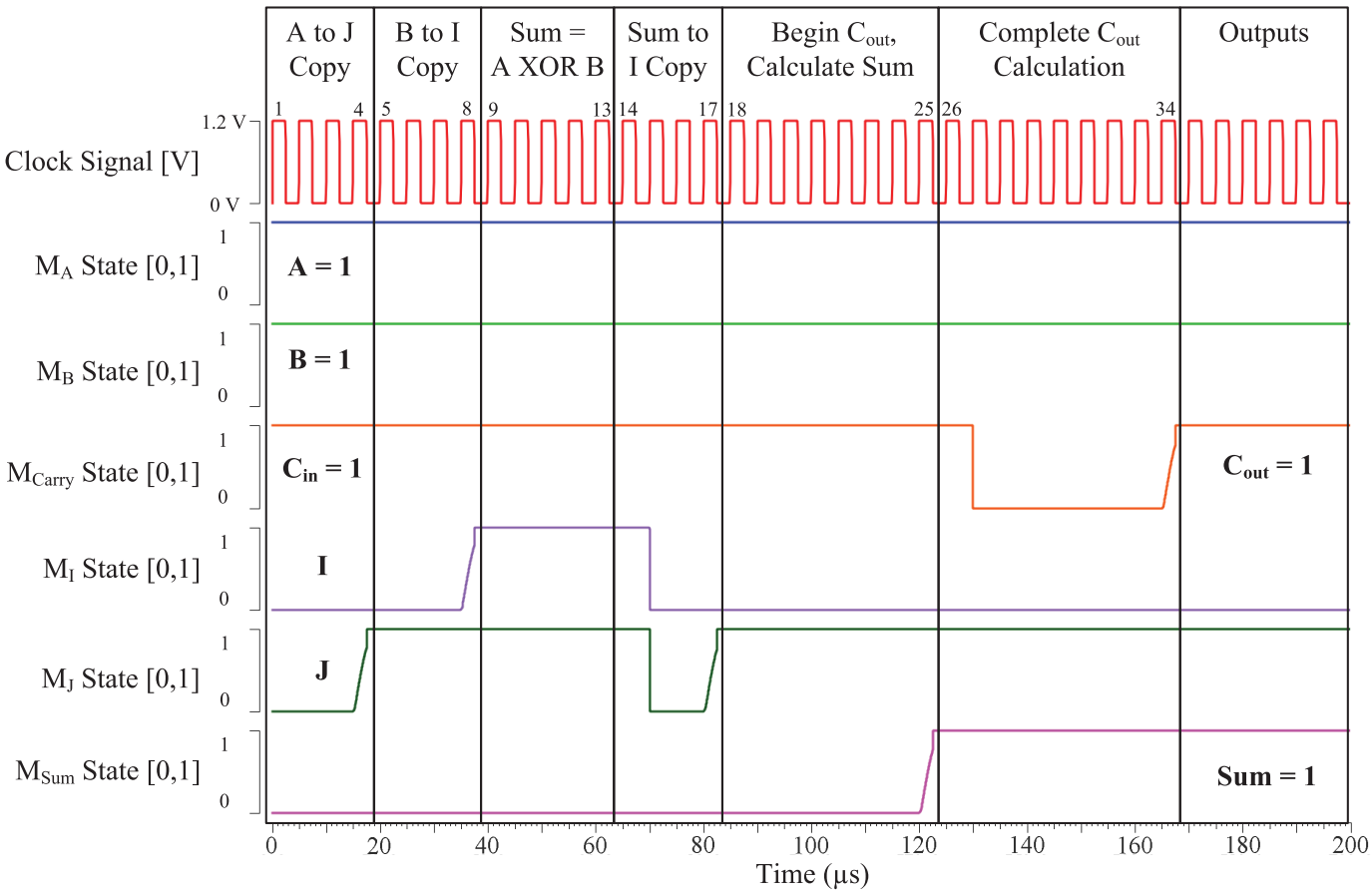


Fig. 8. Simulation of the one-bit full adder using memristors. Six memristors A, B, Carry, M1, M2, and Sum are utilized to perform the function where A, B, and Carry are the inputs and Carry and Sum are the outputs. Each memristor is driven by three transistors for applying V_{COND}, V_{SET}, and V_{RESET}, with the gate voltages not shown here. The states for each of the six memristors are shown throughout the entire calculation. Each set of steps highlights a sub-function that is utilized in the sequence to calculate the final C_{out} and Sum. This set of steps is adapted from [4] to perform the one-bit full adder with the fewest memristors.

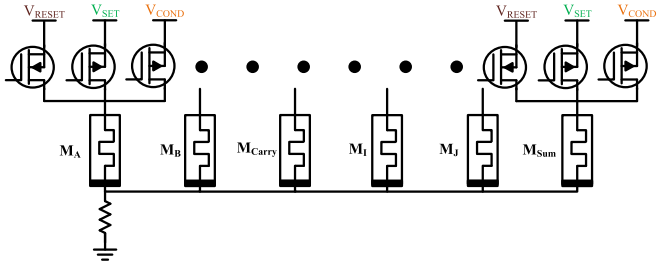


Fig. 9. Memristor structure for implementing a one-bit full adder based on the serial step method provided in [18].

The most complex case of the one-bit full adder is used, where A, B, and C_{in} all begin in the low resistance state (1). The standard logic functions of which the full adder is comprised are separated and highlighted. Overall, the simulation takes 34 steps rather than 29 steps, as buffer steps are required whenever a Reset operation is performed. The results demonstrate proper functionality with the C_{out} and Sum ultimately in the low resistance state (1).

The memristor one-bit full adder requires 427 supporting transistors, 14 times more than the number of transistors required for a standard CMOS implementation. Even if the control circuitry were further optimized, the system would still require many more transistors than required in a pure CMOS implementation. In addition to the large overhead required, the 34 serial steps required to implement this function with only six memristors results in a calculation time of 200 μs. While memristors can perform logic-in-memory, the increase in the delay for performing even simple calculations is far more problematic than the delay resulting from the von Neumann bottleneck. Even if the true devices and simulation were significantly faster, a CMOS implementation would still be multiple orders of magnitude faster than a memristor-based solution.

D. Analysis Of Power-Delay Product

As highlighted previously, the state-change operation of memristors require both a significant amount of power consumption, due to the current required to mobilize the ions, and time, due to the mechanical switching and number of serial steps. A standard figure of merit, the power-delay product (PDP), is measured in simulation and compared to a standard CMOS implementation. The PDP is measured for both the memristor logic and the CMOS logic by measuring the current through each device and calculating the energy

TABLE VI
POWER-DELAY PRODUCT FOR MEMRISTORS AND CMOS

Function	Memristor PDP (J)	CMOS PDP (J)
NAND	$1.29 * 10^{-8}$	$3.74 * 10^{-19}$
AND	$1.58 * 10^{-8}$	$7.17 * 10^{-19}$
NOR	$8.10 * 10^{-9}$	$5.06 * 10^{-19}$
OR	$1.90 * 10^{-8}$	$8.91 * 10^{-19}$
XOR	$1.04 * 10^{-7}$	$1.86 * 10^{-18}$
XNOR	$9.29 * 10^{-8}$	$1.83 * 10^{-18}$
Full Adder	$1.14 * 10^{-7}$	$6.63 * 10^{-18}$

TABLE VII
ENERGY-DELAY PRODUCT FOR MEMRISTORS AND CMOS

Function	Memristor EDP (J·s)	CMOS EDP (J·s)
NAND	$2.58 * 10^{-13}$	$1.87 * 10^{-24}$
AND	$3.95 * 10^{-13}$	$3.59 * 10^{-24}$
NOR	$1.22 * 10^{-13}$	$2.53 * 10^{-24}$
OR	$3.80 * 10^{-13}$	$4.46 * 10^{-24}$
XOR	$7.25 * 10^{-12}$	$9.31 * 10^{-24}$
XNOR	$6.04 * 10^{-12}$	$9.15 * 10^{-24}$
Full Adder	$1.93 * 10^{-11}$	$3.32 * 10^{-23}$

delay product, which is then used with the propagation delay to determine the power delay product. The PDP and EDP are measured for each standard Boolean function in addition to the one-bit full adder, and the results are provided in Tables VI and VII, respectively.

It can be seen that the memristor implementation is generally ten orders of magnitude less efficient than CMOS. Any potential advantage that memristors provide is dwarfed by this unacceptable PDP. This result confirms that the memristor-based logic cannot compete with current CMOS technologies. Even if the control system is further optimized by reducing the overall number of devices, and the memristors are significantly faster, these improvements would still not allow the memristor-based system to compete with a CMOS structure. Faster or lower energy memristors [46] will shrink the gap between these PDP values, but more CMOS transistors are required for the overhead circuit than in a conventional CMOS implementation. Therefore the stateful memristor logic implementation can never be superior to a pure CMOS implementation within a conventional von Neumann architecture.

V. COMPUTING IMPLICATIONS

While previous research posited that the total number of devices can be reduced when utilizing memristors as compared to a CMOS architecture, these research findings typically utilize a large number of serial steps to perform the same function [18]. The large number of serial steps paired with the slow response time of memristors imposes an extraordinary limitation on computation speed and PDP. Moreover, much previous work fails to investigate a potentially critical barrier to the adoption of memristors in computing architectures: the overhead of the required supporting structures.

Further investigation into the required CMOS overhead circuits show that memristors cannot effectively reduce the number of devices required to perform computations. While on the surface it may seem that a one-bit full adder can be performed with only six memristors, this does not take into account the required supporting circuitry, which far surpasses the number of devices with which the same function can be performed in CMOS.

Though the analysis shown here may not describe a fully optimized solution for a control system for memristor-based logic, this control system is simpler than what is truly required for memristor-based computing [30] and what is highlighted in [45]. The system proposed in [45] requires significantly more CMOS transistors, as the complexity scales with the flexibility of the architecture as well as the number of unique voltages that are applied to each memristor. The analysis shown here also does not specifically include the required structures for reading the state of a particular memristor. Reading the state of any device in the architecture requires even more circuitry for each memristor. Converting the state of a memristor into a voltage requires circuits such as sense amplifiers, as depicted in Fig. 3(a) of [30], further increasing the size of the peripheral architecture for a memristor-based logic array.

It is possible to increase the speed of computation when using memristors by decreasing the number of serial steps required; however, this comes at the cost of additional parallel memristors. Moreover, adding memristors increases the number of power switches required, necessitating additional memory and supporting structures. There is a fundamental tradeoff between the number of devices and the speed of operation, and this analysis shows that even with the slowest operation of the full-adder, the required number of devices far surpasses those required for a standard CMOS implementation. Additionally, even if the speed were improved, this improvement must be of multiple orders of magnitude in order to compete with a CMOS implementation. Due to the mechanical switching of a memristor, the speed of operation of a memristor simply cannot compete with a CMOS transistor, causing the memristor-based approach to have a significantly inferior PDP. Even with a significantly faster memristor [46], the overhead requirements are unchanged, requiring more CMOS transistors than a pure CMOS implementation and providing less efficiency.

VI. CONCLUSION

While memristor technology is promising, memristors as computing blocks are shown here to exhibit many shortcomings. The required serial steps and the slow memristor switching speeds coupled with the energy consumption of the overhead circuit lead to a significant increase in PDP, with a result ten orders of magnitude inferior to an equivalent CMOS implementation. With such a wide gap in PDP between memristors and CMOS, even with significant improvements in the control circuitry and the speed of operation of memristors, the memristor-based system simply cannot compete. Furthermore, without even including the required circuitry for

reading the state of the memristors, it has been determined that the number of supporting transistors can be upwards of 50 times the number of transistors required for a standard CMOS implementation. Therefore, though stateful memristor logic may have utility within a non-von Neumann computing architecture that makes efficient use of the memristor non-volatility, the prospects for stateful memristor logic within a conventional computer architecture are severely limited by the overhead circuit requirements.

ACKNOWLEDGMENT

The authors would like to thank M. Rathna, Y. Liu, P. X. Francis, E. Linn, A. Siemon, S. Menzel, and S. Kvatinksy for their contributions toward this paper.

REFERENCES

[1] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. S. P. Wong, "Device scaling limits of Si MOSFETs and their application dependencies," *Proc. IEEE*, vol. 89, no. 3, pp. 259–287, Mar. 2001.

[2] S. Matsunaga *et al.*, "Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions," *Appl. Phys. Express*, vol. 1, no. 9, p. 091301, Aug. 2008.

[3] M. T. Niemier *et al.*, "Nanomagnet logic: Progress toward system-level integration," *J. Phys., Condens. Matter*, vol. 23, no. 49, p. 493202, Nov. 2011.

[4] K. A. Omari and T. J. Hayward, "Chirality-based vortex domain-wall logic gates," *Phys. Rev. Appl.*, vol. 2, no. 4, p. 044001, Oct. 2014.

[5] J. S. Friedman, B. W. Wessels, G. Memik, and A. V. Sahakian, "Emitter-coupled spin-transistor logic: Cascaded spintronic computing beyond 10 GHz," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 5, no. 1, pp. 17–27, Mar. 2015.

[6] D. A. Allwood, G. Xiong, C. C. Faulkner, D. Atkinson, D. Petit, and R. P. Cowburn, "Magnetic domain-wall logic," *Science*, vol. 309, pp. 1688–1692, Sep. 2005.

[7] A. Imre, G. Csaba, L. Ji, A. Orlov, G. H. Bernstein, and W. Porod, "Majority logic gate for magnetic quantum-dot cellular automata," *Science*, vol. 311, no. 5758, pp. 205–208, 2006.

[8] J. A. Currivan, Y. Jang, M. D. Mascaro, M. A. Baldo, and C. A. Ross, "Low energy magnetic domain wall logic in short, narrow, ferromagnetic wires," *IEEE Magn. Lett.*, vol. 3, Apr. 2012, Art. no. 3000104.

[9] J. S. Friedman and A. V. Sahakian, "Complementary magnetic tunnel junction logic," *IEEE Trans. Electron Devices*, vol. 61, no. 4, pp. 1207–1210, Apr. 2014.

[10] J. S. Friedman *et al.*, "Cascaded spintronic logic with low-dimensional carbon," *Nature Commun.*, vol. 8, Jun. 2017, Art. no. 15635.

[11] J. S. Friedman, A. Godkin, A. Henning, Y. Vaknin, Y. Rosenwaks, and A. V. Sahakian, "Threshold logic with electrostatically formed nanowires," *IEEE Trans. Electron Devices*, vol. 63, no. 3, pp. 1388–1391, Mar. 2016.

[12] Y.-M. Lin, J. Appenzeller, J. Knoch, and P. Avouris, "High-performance carbon nanotube field-effect transistor with tunable polarities," *IEEE Trans. Nanotechnol.*, vol. 4, no. 5, pp. 481–489, Sep. 2005.

[13] M. M. Shulaker *et al.*, "Carbon nanotube computer," *Nature*, vol. 501, no. 7468, pp. 526–530, 2013.

[14] M. H. Ben-Jamaa, K. Mohanram, and G. De Micheli, "An efficient gate library for ambipolar CNTFET logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 242–255, Feb. 2011.

[15] M. Prakash and N. Gershenfeld, "Microfluidic bubble logic," *Science*, vol. 315, no. 5813, pp. 832–835, Feb. 2007.

[16] W. Kang, Z. Wang, Y. Zhang, J.-O. Klein, W. Lv, and W. Zhao, "Spintronic logic design methodology based on spin Hall effect-driven magnetic tunnel junctions," *J. Phys. D, Appl. Phys.*, vol. 49, no. 6, p. 65008, 2016.

[17] H. Zhang, W. Kang, L. Wang, K. L. Wang, and W. Zhao, "Stateful reconfigurable logic via a single-voltage-gated spin Hall-effect driven magnetic tunnel junction in a spintronic memory," *IEEE Trans. Electron Devices*, vol. 64, no. 10, pp. 4295–4301, Oct. 2017.

[18] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[19] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, Apr. 2010.

[20] L. O. Chua, "Memristor—The missing circuit element," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.

[21] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[22] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015.

[23] S. Balatti, S. Ambrogio, and D. Ielmini, "Normally-off logic based on resistive switches—Part I: Logic gates," *IEEE Trans. Electron Devices*, vol. 62, no. 6, pp. 1831–1838, Jun. 2015.

[24] S. Balatti, S. Ambrogio, and D. Ielmini, "Normally-off logic based on resistive switches—Part II: Logic circuits," *IEEE Trans. Electron Devices*, vol. 62, no. 6, pp. 1839–1847, Jun. 2015.

[25] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM NANOARCH*, Jul. 2009, pp. 33–36.

[26] S. Shin, K. Kim, and S.-M. Kang, "Reconfigurable stateful nor gate for large-scale logic-array integrations," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 58, no. 7, pp. 442–446, Jul. 2011.

[27] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Solids Surf.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.

[28] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.

[29] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE ISIT*, Jul. 2013, pp. 156–160.

[30] J. Reuben *et al.*, "Memristive logic: A framework for evaluation and comparison," in *Proc. IEEE PATMOS*, Sep. 2017, pp. 1–8.

[31] K. Miller, K. S. Nalwa, A. Bergerud, N. M. Neihart, and S. Chaudhary, "Memristive behavior in thin anodic titania," *IEEE Electron Device Lett.*, vol. 31, no. 7, pp. 737–739, Jul. 2010.

[32] Q. Xia *et al.*, "Memristor-CMOS hybrid integrated circuits for reconfigurable logic," *Nano Lett.*, vol. 9, no. 10, pp. 3640–3645, 2009.

[33] I. Valov *et al.*, "Nanobatteries in redox-based resistive switches require extension of memristor theory," *Nature Commun.*, vol. 4, Apr. 2013, Art. no. 1771.

[34] H.-S. P. Wong and S. Salahuddin, "Memory leads the way to better computing," *Nature Nanotechnol.*, vol. 10, no. 3, pp. 191–194, Mar. 2015.

[35] A. H. Edwards, H. J. Barnaby, K. A. Campbell, M. N. Kozicki, W. Liu, and M. J. Marinella, "Reconfigurable memristive device technologies," *Proc. IEEE*, vol. 103, no. 7, pp. 1004–1033, Jul. 2015.

[36] I. Valov and M. N. Kozicki, "Cation-based resistance change memory," *J. Phys. D, Appl. Phys.*, vol. 46, no. 8, p. 074005, Jan. 2013.

[37] D. J. Wouters, R. Waser, and M. Wuttig, "Phase-change and redox-based resistive switching memories," *Proc. IEEE*, vol. 103, no. 8, pp. 1274–1288, Aug. 2015.

[38] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL—Memristor ratioed logic," in *Proc. Int. Workshop Cellular Nanosc. Netw. Their Appl.*, 2012, pp. 1–6.

[39] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[40] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Mater.*, vol. 9, no. 5, pp. 403–406, May 2010.

[41] R. Rosezin, E. Linn, C. Kügeler, R. Bruchhaus, and R. Waser, "Crossbar logic using bipolar and complementary resistive switches," *IEEE Electron Device Lett.*, vol. 32, no. 6, pp. 710–712, Jun. 2011.

[42] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann—Logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[43] L. Guckert and E. E. Swartzlander, "MAD gates—Memristor logic design using driver circuitry," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 64, no. 2, pp. 171–175, Feb. 2017.

[44] L. Guckert and E. E. Swartzlander, "Optimized memristor-based multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 2, pp. 373–385, Feb. 2017.

[45] R. Ben Hur and S. Kvatinsky, "Memristive memory processing unit (MPU) controller for in-memory processing," in *Proc. IEEE ICSEE*, Nov. 2016, pp. 1–5.
[46] B. J. Choi *et al.*, "High-speed and low-energy nitride memristors," *Adv. Funct. Mater.*, vol. 26, no. 29, pp. 5290–5296, 2016.

Xuan Hu (S'16) received the B.S. degree in electrical and information engineering from Huaqiao University, Xiamen, China, in 2013, and the M.S. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2015.

He is currently pursuing the Ph.D. degree in electrical engineering with the Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas, Richardson, TX, USA.

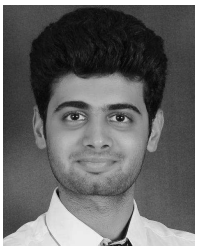
His current research focuses on circuit design and modeling of efficient memristive, spintronic, and carbon nanotube logic circuits.

Michael J. Schultis received the B.E. degree in biomedical and electrical engineering from Vanderbilt University, Nashville, TN, USA, in 2014. He is currently pursuing the M.S. degree in electrical engineering from The University of Texas at Dallas, Richardson, TX, USA, in 2018.

His current research focus is on stateful memristor logic.

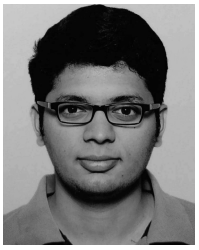
Matthew Kramer received the B.E. degree in Electrical Engineering in 2015.

He is currently pursuing the M.S. degree with the Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas, with a focus on circuits and systems.



Archit Bagla received the B.E. degree in electronics and communication engineering from the Institute of Technology, Nirma University, in 2016.

He is currently pursuing the M.S. degree with the Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas. His current research focuses on analog circuit design with spintronic and emerging technologies.



Akshay Shetty received the B.E. degree in electronics and communication engineering from the NMAM Institute of Technology (autonomous under VTU), Nitte, India, in 2015.

He is currently pursuing the M.S. degree with the Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas.

Joseph S. Friedman (S'09–M'14) received the A.B. and B.E. degrees from Dartmouth College, Hanover, NH, USA, in 2009, and the M.S. and Ph.D. degrees in electrical & computer engineering from Northwestern University, Evanston, IL, USA, in 2010 and 2014, respectively.

He joined The University of Texas at Dallas, Richardson, TX, USA, in 2016, where he is currently an Assistant Professor of electrical & computer engineering and the Director of the NanoSpinCompute Laboratory. From 2014 to 2016, he was a Centre National de la Recherche Scientifique Research Associate with the Institut d'Electronique Fondamentale, Université Paris-Sud, Orsay, France. He has also been a Visiting Professor at Politecnico di Torino, Turin, Italy, a Guest Scientist at RWTH Aachen University, Aachen, Germany, and worked on logic design automation as an intern at Intel Corporation, Santa Clara, CA, USA.

Dr. Friedman is a member of the editorial board of the *Microelectronics Journal*, the technical program committees of DAC, SPIE Spintronics, NANOARCH, GLSVSI, and ICECS, the review committee of ISCAS, and the IEEE Circuits & Systems Society Nanoelectronics and Gigascale Systems Technical Committee. He has also been awarded a Fulbright Postdoctoral Fellowship. His research interests include the invention and design of novel cascaded computing structures based on nanoscale and quantum mechanical phenomena, with particular emphasis on spintronics.