INVITED PAPER

Leveraging Memristive Systems in the Construction of Digital Logic Circuits

The authors discuss advantages of using memristors in reprogrammable digital systems to replace bulky SRAM-based configurable memories.

By Garrett S. Rose, *Member IEEE*, Jeyavijayan Rajendran, *Student Member IEEE*, Harika Manem, *Student Member IEEE*, Ramesh Karri, *Member IEEE*, and Robinson E. Pino, *Senior Member IEEE*

ABSTRACT | The recent emergence of the memristor has led to a great deal of research into the potential uses of the devices. Specifically, the innate reconfigurability of memristors can be exploited for applications ranging from multilevel memory, programmable logic, and neuromorphic computing, to name a few. In this work, memristors are explored for their potential use in dense programmable logic circuits. While much of the work is focused on Boolean logic, nontraditional styles including threshold logic and neuromorhpic computing are also considered. In addition to an analysis of the circuits themselves, computer-aided design (CAD) flows are presented which have been used to map digital logic functionality to dense complementary metal–oxide–semiconductor (CMOS)-memristive logic arrays. As exemplified through the circuits described here memristor-based digital logic holds great potential for high-density and energy-efficient computing.

KEYWORDS | Digital integrated circuits; memristors; nanoelectronics; programmable circuits

Manuscript received October 29, 2010; revised May 6, 2011; accepted August 20, 2011. Date of publication October 21, 2011; date of current version May 10, 2012. The contractors acknowledge government support in the publication of this paper. This material is based upon work funded by the U.S. Air Force Research Laboratory (AFRL) under Contract FA8750-09-2-0157. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL.

G. S. Rose and **R. E. Pino** are with the Information Directorate, Air Force Research Laboratory, Rome, NY 13441 USA (e-mail: garrett.rose@rl.af.mil; robinson.pino@rl.af.mil).

J. Rajendran, **H. Manem**, and **R. Karri** are with the Department of Electrical and Computer Engineering, Polytechnic Institute of New York University, Brooklyn, NY 11201 USA (e-mail: jrajen01@students.poly.edu; hmanem01@students.poly.edu; rkarri@duke.poly.edu).

Digital Object Identifier: 10.1109/JPROC.2011.2167489

I. INTRODUCTION

Emerging nanoelectronic technologies have enabled the implementation of several logic and memory architectures with greater device densities and better energy-delay metrics [1]–[3]. The discovery of memristance [4], [5] has further broadened the scope of hybrid complementary metal–oxide–semiconductor (CMOS)/nano architectures to implement non-Boolean logic. Memristance is a property that relates charge and flux leading to a device behavior where the resistance changes with an applied electric field [4]. This reconfigurability in memristance can be leveraged for a wide variety of applications ranging from neuromorphic computing to multilevel logic and memory systems. Memristive synapses have been interfaced with CMOS neurons in [7] and [8] and a cellular nonlinear network based on memristors was proposed in [9]. Memristors were used to build nonvolatile two-level and multilevel memories in [10] and [11], respectively. Memristive devices can also be used in digital logic as programmable switches in switching blocks [12] and to build block memories [13]. This work considers the versatility and scope of memristors and memristive systems when employed in several nanoelectronic and hybrid CMOS/nano circuits to realize several flavors of logic implementations from Boolean to threshold logic.

The memristor, or "memory resistor," was first theorized by Leon Chua in 1971 [4]. Only theoretical for more than 30 years, researchers at Hewlett-Packard recently announced the discovery of memristors fabricated in their lab [5]. In terms of its behavior, a memristor is a device whose resistance changes under given toggle conditions (e.g., exceeding some voltage) and then holds that resistance until another toggle condition is met. In this

way, memristors can be thought of as reconfigurable resistors with memory. Later work by Chua extends the concept of the memristor to what is known as a memristive system, a more complex device whose memristance depends on several state variables [14]. Most memristive devices fabricated to date are actually memristive systems as they exhibit a nonlinear dependency on electric field [15]. The device models developed in this work reflect this reality by modeling these devices as memristive systems.

This paper presents different methodologies in which memristive devices can be leveraged to realize conventional (Boolean) and nonconventional (neural, threshold) logic applications. This analysis begins with the usage of memristive switches in a basic programmable logic array (PLA) circuit to implement Boolean logic. PLA-based circuits have been in existence for a few decades but have gained recent popularity in the design of nanoscale structures due to their high logic density and potentially straightforward implementation [2]. For the PLA circuits considered here memristive switches are used simply as bistable switches that can be inserted at the crosspoints of a nanoscale crossbar array [16] to implement an AND–OR-style PLA. The reconfigurable nature of memristance allows the switches to be turned "ON" or "OFF" to change the logic mapping, allowing for reprogrammability in a memristive PLA. While very dense and potentially energy-delay efficient, memristive PLA circuits suffer from signal degradation after each logic stage.

Boolean logic can also be implemented with majority gates built from memristors or memristive systems in conjunction with a particularly interesting circuit that can be used for signal restoration at the nanoscale, the Goto pair [17], [18]. The basic premise of the majority gate is that the output of the gate takes on the same value as the majority of its inputs. In this case, memristive devices are used as bistable switches, as in the memristive PLA, and the input signals are fed into Goto pair circuits. The Goto pair latches onto the majority of the input signals provided by the memristive switches at its input, thus implementing majority logic. Since memristors and memristive systems are reconfigurable, the circuit implemented is essentially a memristive programmable majority logic array (PMLA) [19]. This work details the design of such a memristive PMLA circuit and how it can be configured to implement Boolean logic.

Another attractive application detailed in this paper uses the innate behavior of the variable resistance in memristors and memristive systems to function as synapses in simple neural networks. The approach taken for the development of such a neuromorphic system is in the initial design of a typical neural network as a threshold function based on the weighted sum of the inputs [8], [9]. A threshold gate based on such a function can be configured to implement both neuromorphic logic and Boolean logic. It is worth noting that in the aforementioned memristive PMLA if the memristive switches were to be used in several states, as opposed to just "ON" and "OFF," the gate would then implement threshold logic functionality. This work illustrates such a threshold gate built exclusively from nanoscale elements (memristors and Goto pairs) and the manner in which it can be configured for neuromorphic computing. The potential of such memristive neuromorphic circuits is demonstrated through a basic image recognition example.

A significant concern for the aforementioned memristive majority and threshold array structures is the necessity for negative differential resistance (NDR) devices [17], [18] for their realization. NDR devices, while attractive for their small size, may not be so easy to integrate and are not easily available for metal–oxide-based fabrication. Thus, hybrid CMOS-memristive threshold logic gates are proposed which can be readily integrated with existing CMOS technologies. Two flavors of hybrid CMOS/nano threshold gates are considered: a current mirror threshold gate (CMTG) and a charge sharing threshold gate (CSTG). Simulation results demonstrating the potential for each of these CMOS-memristive logic circuits are also provided. Table 1 illustrates the attributes and requirements for the different memristive logic styles discussed. The table lists the type of logic implemented by the memristive circuit, availability of signal restoration, area, energy, and delay. The proposed nano and hybid CMOS/nano logic circuits are also matched against CMOS-based threshold and Boolean architectures, specifically capacitive threshold logic (CTL) gates [19] and lookup table (LUT)-based field-programmable gate arrays (FPGAs) for the aforementioned parameters.

Table 1 Attributes of Memristive Logic Styles

Circuit	Logic	Sig. Rest.	Tech.	Energy	Delay (s)
PLA	Boolean	No	Nano	Low	$\sim 10^{-8}$
PMLA	Majority	Yes	Nano	Low	$\sim 10^{-9}$
TLA	Threshold	Yes	Nano	Low	$\sim 10^{-9}$
CSTG	Threshold	Yes	Hybrid	Low	$\sim 10^{-5}$
CMTG	Threshold	Yes	Hybrid	Low	$\sim 10^{-5}$
CTL*	Threshold	Yes	CMOS	High	$\sim 10^{-5}$
LUT*	Boolean	Yes	CMOS	High	$\sim 10^{-9}$

* Non-memristive logic style circuits.

Another key contribution of this work is the development of a design flow for the logic synthesis of memristive threshold logic. A bottom-up CAD framework is developed to map the weights of threshold functions to particular memristance values, synthesize a netlist using memristive threshold logic gates, and evaluate the performance of complete logic systems. The weights for the proposed hybrid CMOS/nano threshold gates are computed in terms of memristance and the gates are subsequently evaluated with respect to performance, area, and power metrics. The Berkeley SIS synthesis tool [20] is then used for Boolean logic synthesis of the characterized threshold gates and a netlist is generated in terms of the memristive threshold logic gates. Performance and area comparison between the proposed memristive logic gates, capacitive threshold logic (CTL), and LUT-based FPGAs are provided using the ISCAS-85 benchmarking circuits first introduced at the 1985 International Symposium on Circuits and Systems [32].

The remainder of the paper is divided as follows. Section II details the device model for the memristive device used in this work. Section III illustrates some basic nanoelectronic circuits and the method in which they can be used to implement a memristive PLA. Section IV describes the memristive PMLA circuit used to implement Boolean logic and extends it to the implementation of threshold logic that can be used for neural-network-based applications. Two hybrid CMOS/nano threshold gates, the CMTG and the CSTG, are explained in Section V. Section VI describes the CAD framework for mapping Boolean logic to hybrid CMOS-memristive threshold logic gates. Finally, some concluding remarks and discussion are provided in Section VII.

II. MEMRISTANCE AND COMPACT DEVICE MODELING

A memristive system model is developed based on TiO_2 memristive devices using the two resistor model presented by HP Labs [6]. This model assumes two series resistors (R_{on} and R_{off}) that represent doped and undoped layers of TiO_2. The boundary between the regions (w) moves between 0 and D (the thickness of the active layer) as a function of an applied electric field

$$M(w) = \frac{w}{D}R_{on} + \left(1 - \frac{w}{D}\right)R_{off}. \tag{1}$$

The memristance as a function of charge (q) and flux (φ) has been derived in prior work based on the fact that the boundary (w) moves from 0 to D as ions drift from the doped to the undoped layer [5]. For simplicity, most models have assumed a linear ionic drift velocity $v = \mu E$ where μ is the mobility of the ionic dopants and E is the applied electric field. However, it has been pointed out

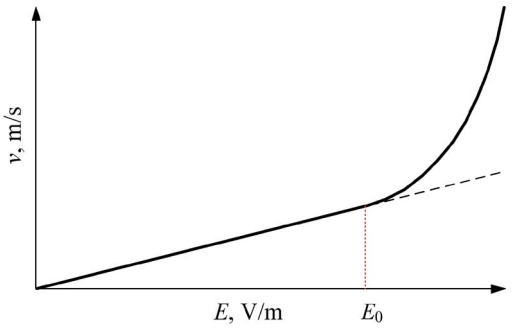

Fig. 1. *Exponential drift (bold) and linear drift (dashed) in memristors.*

that the drift velocity of ions within some memristors is actually nonlinear and even exponential [15]. As pointed out in [15], the exponential drift velocity can be modeled as described by (2). This exponential drift velocity is shown in Fig. 1 where the velocity is plotted as function of the applied electric field E

$$v = \begin{cases} \mu E, & E \ll E_0 \\ \mu E_0 \exp(E/E_0), & E \sim E_0. \end{cases} \tag{2}$$

In order to model a memristor (more specifically, a memristive system) with the exponential drift velocity we assume the ionic dopant mobility is exponential with the applied electric field. In this work, the memristor model is developed based on the following relationships for the mobility:

$$\mu = \begin{cases} \dfrac{\mu_z V_{th,pos}}{D} \cdot \exp\left(\dfrac{V_{appl}}{V_{th,pos}}\right), & \text{if } V_{appl} \geq V_{th,pos} \\ \mu_z, & \text{if } V_{th,pos} > V_{appl} > V_{th,neg} \\ \dfrac{\mu_z |V_{th,neg}|}{D} \cdot \exp\left(\dfrac{V_{appl}}{V_{th,neg}}\right), & \text{if } V_{appl} \leq V_{th,neg} \end{cases} \tag{3}$$

where μ_z is the mobility for small electric fields, M is the present memristance, V_{appl} is the applied voltage (directly proportional to E), $V_{th,pos}$ is the positive threshold voltage, $V_{th,neg}$ is the negative threshold voltage, and D is the thickness of the device. This new mobility is used with the electric field to determine the drift velocity similar to the linear drift velocity model. In this context, however, nonlinear drift results from variable mobility.

Using the variable mobility μ the equation for the memristance M as a function of flux φ remains the same as

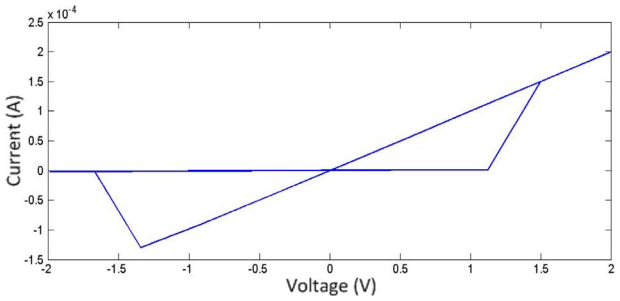

Fig. 2. *The current versus voltage characteristics of a memristor exhibiting exponential drift beyond ±1.3 V ($E_0 \sim 25$ MV/m).*

that described in [6]

$$M(t) = R_0\sqrt{\left(1 - \frac{2\eta\Delta R\varphi(t)}{Q_0 R_0^2}\right)} \tag{4}$$

where R_0 is the maximum resistance ($R_0 \approx R_{\text{off}}$), Q_0 is the charge required for w to migrate from 0 to D, ΔR is the difference between R_{off} and R_{on}, and η (either $+1$ or -1) is the polarity of the applied voltage signal. The flux $\varphi(t)$ is simply the integral of the applied voltage over the entire usage history of the device $\int V_{\text{appl}}(t)dt$ such that the memristance state contains information of that history. In (4), the variable mobility term μ is contained within the charge term $Q_0 = D^2/(\mu R_{\text{on}})$.

The variable mobility model is included as a Verilog-A model for circuit simulations using the Cadence Spectre simulation engine. Fig. 2 shows a plot from Cadence Spectre for the current versus voltage response of the memristive device using the variable mobility model described here. For the device considered, R_{on} is 121 kΩ, R_{off} is 121 MΩ, D is 50 nm, μ_0 is 3×10^{-18} m^2/V · s and $E_0 \sim$ 25 MV/m. The maximum value of φ is 10^6 V · s. It is to be noted that for an operating voltage less than ±1.3 V (threshold voltage for $D = 50$ nm and $E_0 \sim 25$ MV/m) the memristor follows the linear drift model and the device memristance does not alter much with respect to time (almost constant), as seen in Fig. 2. So in the functional mode, when the circuit is being used with an operating voltage around 1 V, the memristance is essentially constant. During the programming mode, where the memristance is set, programming voltages greater than 1.3 V (exponential drift) are used to set the memristance.

III. BASICS OF NANOELECTRONIC LOGIC CIRCUITS

In recent years, several approaches have been proposed for integrating nanoscale device technologies into nanoelectronic circuitry. Many of the structures considered for nanoelectronic logic and memory circuits can be constructed using memristors as they are typically nanoscale devices. More specifically, a great deal of nanoelectronic circuit research has focused on structures such as the crossbar array [16]. These crossbar array structures are denser and serve as a good starting point for digital memristive circuits.

A nanoscale crossbar array consists of two sets of nanowires running perpendicular to one another [16]. At the intersection of each set of two perpendicular nanowires a memristive device can be inserted. Many nanoscale circuits devised so far have employed this array for realizing logic and memory [1], [2]. Fig. 3 illustrates a typical crossbar array with memristive switches sandwiched between the nanowires. This circuit structure is attractive for several reasons including high device density and simplicity in design.

One drawback with the crossbar array is the lack of inverting functionality and signal restoration. A second, perhaps more critical issue with crossbar arrays constructed with resistive or memristive switches, such as those considered here, is that of sneak path currents [16]. To illustrate the issue of sneak path currents in a crossbar, first consider an ideal array consisting of switches that are diode like in nature such that current flows in one direction only. In such an ideal array the diode-like structures would ensure that the output depends only on the selected or ON-path devices, as it should. However, most memristive devices and other resistive devices are bidirectional. When these devices are used in a crossbar array they will allow current to flow through unselected devices such that the output might depend on the OFF-path devices as much as the ON-path devices. The problem can be somewhat mitigated by grounding unselected rows and/or columns [22], but ultimately the issue of sneak path currents will drastically limit the size of the array itself.

These issues notwithstanding, the crossbar array is a useful starting point as we consider memristive logic circuits, due in part to the prevelant use of such structures in nanoelectronic designs [1], [2], [14]. Furthermore, the crossbar array can be very reliable if the size of the array is

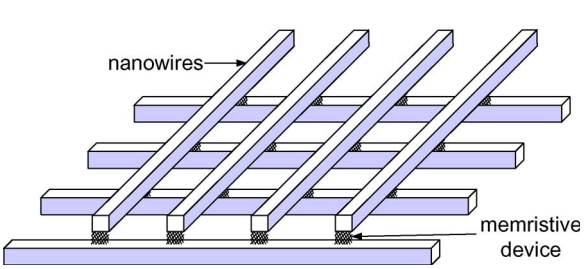

Fig. 3. *A nanoscale crossbar array composed of nanowires crisscrossing perpendicularly with memristive devices at each crosspoint [13].*

limited. A reasonable use of small crossbar PLA structures might be as part of a configurable logic block within a larger reconfigurable system. With these issues and possibilities in mind an exploration of a memristive crossbar-array-based PLA is included here.

A. Memristive Programmable Logic Arrays (PLAs)

Given the switching nature of nanoelectronic devices, including memristors, it is possible to construct dense PLAs using nanoscale crossbar structures like that shown in Fig. 3. In the past, PLAs were popularly used for implementing combinational logic. They are less frequently used now as modern logic synthesis often produces faster circuits. However, they have been resurrected for implementation in nanoscale structures for their high logic density and potentially straightforward implementation.

Any logic function can be expressed in sum-of-products form where each output is determined by applying an OR (logical union or sum) to several AND (logical intersect or product) terms, each determined from a subset of the inputs. A PLA consists of an AND plane to compute the minterms and an OR plane to compute the final outputs. Fig. 4 shows a basic representation of a PLA. Here each dark blue memristor is considered to be configured to the memristance value R_{on}. Likewise, the light blue memristors are configured to be R_{off} such that they can be considered open circuits. As seen in the AND plane, connections are made between columns (inputs) and rows (minterms) that determine the various AND terms required for the Boolean function. For example, the bottom row in the AND plane is connected to input columns *A*, *B*, and *C*, such that the minterm is *ABC*. The OR plane operates in a similar fashion with rows connected to columns (outputs) to implement the OR terms.

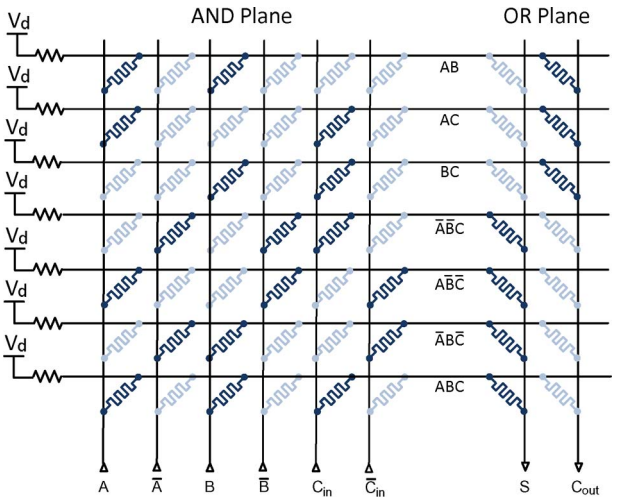

Fig. 4. *Representation of a PLA where the darker devices represent connections ("ON" memristors) while a lighter device signifies a disconnected crosspoint ("OFF" memristors).*

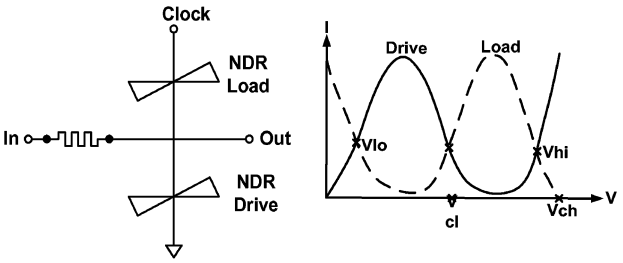

Fig. 5. *Schematic of a Goto pair circuit (left) along with its load line diagram (right) [18].*

If memristors are used simply as two state switches then they can be inserted at the crosspoints of a crossbar array to implement an AND–OR-style PLA like that shown in Fig. 4. Memristors and memristive systems also offer the important characteristic of reconfigurability so that logic can be mapped to the array by switching each memristor either on or off. Furthermore, if a change is required, then the original mapping can be easily updated as the memristance can be reconfigured from on to off or *vice versa* as desired. Of course, using memristive devices in this way neglects all of the possible memristance states between R_{on} and R_{off}. As will be discussed later in this work, the ability to configure memristors within a range of possible memristance states can be leveraged in the realization of threshold logic using PLA-like circuits.

B. Signal Restoration and NDR

One issue with PLA-like circuits is that they suffer from the issue of signal degradation as each connected crosspoint (memristor or otherwise) incurs a voltage drop. Furthermore, since a single AND–OR PLA only constitutes two levels of logic many such arrays must be cascaded together in order to implement complex logic circuits. After propagating through several levels of logic a voltage signal meant to represent logic 1 may become so degraded that it is indistinguishable from logic 0. Thus, buffers, latches, or some other circuits must be placed at the outputs of the PLAs in order to restore the voltage signals.

A particularly interesting circuit that can be used for signal restoration at the nanoscale is the Goto pair, shown in Fig. 5 [17], [18]. This circuit consists of two NDR devices (usually resonant tunneling diodes), called the *load* and *drive*, respectively, in series leading to three operating points, two stable and one unstable. The instability of the middle operating point is due to the fact that both devices are biased in the NDR region [17]. Hence, only the two stable points of this circuit can be used to represent and store data as a voltage difference between the two NDR devices. The Goto pair can be used as a flip-flop that is designed to operate on one of the two stable points based on the input. This is accomplished by applying current

through the input node (shown in Fig. 5) as the clock transitions from a low to high value. If the input current is large enough, then the current through the circuit is effectively raised which forces the circuit to operate at the high voltage operating point (VHI). Similarly, a negative input current will pull the circuit towards the low voltage operating point (VLO). Further details for how this circuit operates can be obtained from [18] and [21].

The Goto pair, and NDR devices in general, has been considered for several nanoscale logic circuits since it provides signal restoration using only two-terminal devices. This is important because even in a hybrid CMOS/nano system using CMOS for signal restoration increases the area [22], [23]. The Goto pair is one circuit that makes it possible to compute many stages of logic completely within nanoscale circuitry. An example architectural proposal that utilizes the Goto pair is the NanoFabric designed by Seth Goldstein *et al.* [24]. In the NanoFabric, Goto pair circuits are placed on the outputs of molecular crossbar arrays to implement AND–OR logic. The NanoFabric does not alter the implementation of the logic but increases logic depth through the utilization of the Goto pair.

However, as demonstrated in [21], the Goto pair can easily be used as a majority gate if multiple lines drive its input. A majority gate is a particular class of threshold logic gate where the output is high if a majority of its inputs are high and low otherwise. Interestingly, if one input of the three-input majority gate is held low, the gate behaves like an AND gate. Likewise, if one input is held high the gate behaves like an OR gate. Digital circuits can be designed using majority gates as either AND or OR gates or, as described in [18] and [25], the majority gate functionality can be utilized to minimize the number of gates needed. This cost reduction in terms of area when using majority logic, or more generally threshold logic, is an example of the intrinsic versatility of such gates.

C. Other Considerations

While crossbar arrays are attractive in terms of logic density, nanoscale CMOS can help improve performance with only a small area penalty. Furthermore, the inclusion of CMOS throughout a hybrid CMOS-memristive logic circuit can be a useful alternative to nanoscale circuits such as the Goto pair for signal restoration. This is particularly important in the near term where nanoscale NDR devices are hard to realize while memristors are successfully fabricated by several researchers. In this work, we consider logic circuits based on dense memristive crossbar arrays and NDR-based circuits (i.e., Goto pairs) only as an example. In the near term, it is believed that hybrid CMOS-memristive circuits, where memristors and CMOS transistors can coexist at all levels of design, offer many advantages to pure CMOS logic while also providing a realizable platform for implementation [1], [26].

IV. THRESHOLD LOGIC AND MEMRISTIVE SYNAPSES

As described earlier, memristive devices are essentially defined by a variable resistance (or memristance) which varies with the application of an electric field across the device. Such reconfigurable devices can be harnessed for several applications including memory and reconfigurable logic. The innate behavior of variable resistance is similar to the operation of a synapse found in neural networks. More specifically, a typical neural network can be modeled as a threshold function based on the weighted sum of the inputs

$$Y = \begin{cases} 0, & \text{if } \sum w_i x_i < T \\ 1, & \text{if } \sum w_i x_i \geq T \end{cases} \tag{5}$$

where x_i are the inputs, w_i are weights for respective inputs x_i, and T is the threshold.

The function described by (5) can be broken into at least three basic pieces: 1) the multiplication of weights and inputs; 2) the summation of all weighted inputs; and 3) a comparison to the threshold to determine the binary output of the function. A synapse is essentially defined by the multiplication function where the inputs are weighed by some given value. For the memristive circuits considered, memristors are used to implement the synapse where a voltage input is multiplied by the inverse of the memristance (the weight). Since this leads to a current which represents the weighted input the summation circuit can easily be implemented via Kirchoff's Law by summing the currents at a common node. Alternatively, as will be described, charge sharing can be utilized for the implementation of a low-power summation circuit. The third part to (5), the comparison, can be implemented using a CMOS comparator or buffer or even a Goto pair.

A. Majority Logic

Before considering how memristors can be used to implement threshold logic it is worthwhile to first consider circuits that are useful for majority logic. A majority logic gate is simply defined as a gate whose output takes on the same value as the majority of its inputs. If a majority of the inputs are logic 1, then the output is logic 1 and logic 0 otherwise. Furthermore, majority logic is actually a subset of threshold logic where (5) still holds true with the constraint that the inputs are all binary and the weights are all equal. Memristors can be used as the weights for a majority logic gate but they must be configured to either R_{on} or R_{off} and nothing in between. Such a configuration assumes R_{off} is essentially an open circuit and R_{on} represents the common weight required for majority logic operation.

The majority logic circuit considered here uses memristance as the weight. Specifically, a voltage input signal

applied across a memristor leads to a current representing the weighted input. Since currents can be easily summed by connecting wires at a common node the summation component of the majority logic gate is accomplished via Kirchoff's Current Law. Finally, a comparison to a threshold value can be performed using either a current comparator (discussed in Section V of threshold logic) or a Goto pair circuit. In fact, the Goto pair is an attractive candidate since it can be implemented using all nanoscale components. That said, we will also consider hybrid CMOS/nano options later as the NDR devices required for the Goto pair can be difficult to fabricate.

Assuming the fabrication process allows for the construction of Goto pair circuits, they could be constructed along with memristors to implement a dense majority logic array. Fig. 6 illustrates such an array called as a PMLA due to the reconfigurable nature of the memristors themselves [4], [18]. As can be seen in the figure, memristors are connected to individual inputs on one end and the input of a Goto pair (latch in the figure) on the other. The common wire that connects the memristors to the Goto pair is the node at which the currents representing the weighted inputs are summed together. This summed current (weighted sum of the inputs) then drives the Goto pair such that a current greater than the threshold produces a logic 1 and a logic 0 otherwise.

Since the memristance value of a memristor can be adjusted, the weights of a particular majority gate in the PMLA can be tailored to implement different logic functions. For example, consider the upper left quadrant of the PMLA in Fig. 6. As mentioned before, each dark blue memristor is considered to be configured for memristance value R_{on} and the light blue memristors are configured to be R_{off}. The four rows of the quadrant drive a corresponding Goto pair latches L clocked using signal *Clock1*. Since the outputs of the Goto pairs are the four rows labeled $R0$, $R1$, $R2$, and $R3$, the configured functionality of the upper left quadrant is

$$R0 = \mathrm{Maj}(A, B, C_{\mathrm{in}}) \quad R1 = \mathrm{Maj}(\overline{A}, \overline{B}, \overline{C}_{\mathrm{in}})$$
$$R2 = \mathrm{Maj}(A, B, \overline{C}_{\mathrm{in}}) \quad R3 = C_{\mathrm{in}} \tag{6}$$

where $\mathrm{Maj}(x, y, z)$ is the majority function of inputs x, y, and z. As described in [18], the term $R0 = \mathrm{Maj}(A, B, C_{\mathrm{in}})$ is actually the carry out term (C_{out}) of a full adder. Likewise, as shown in Fig. 6, the terms $R1$, $R2$, and $R3$ can be taken together to implement the sum term $S = \mathrm{Maj}(R1, R2, R3)$ such that the configuration shown is the mapping of a one-bit full adder. Simulation results showing how the PMLA functions when mapped to the full adder can be seen in Fig. 7. As can be seen in the figure, the output is valid when *Clock3* is high.

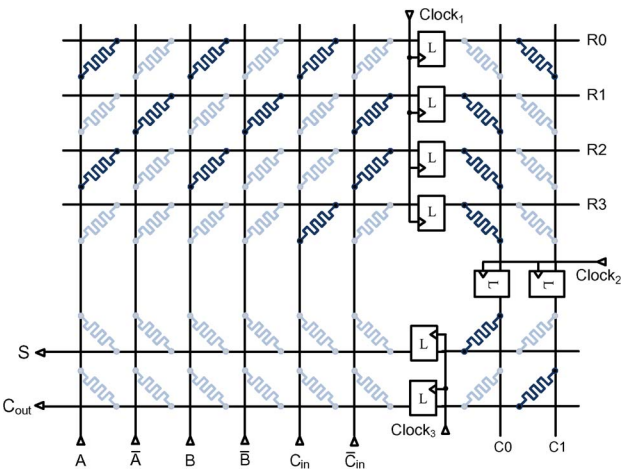

Fig. 6. *Schematic view of the PMLA. Each latch can be implemented using NDR-based Goto pairs [15].*

B. PMLA-Like Threshold Logic Arrays

Threshold logic, and more generally a neural network, is especially useful for applications such as pattern recognition. In a recent analysis of a PMLA-style threshold gate array, the ability of a simple circuit to recognize and distinguish between two images was explored [27]. The actual circuit, shown in Fig. 8, is essentially the same as what has been described for the PMLA. However, unlike the PMLA, in a memristive threshold logic array the memristive devices are configured anywhere within a range of memristance states between R_{on} and R_{off}. For the circuit shown in Fig. 8 the devices are each configured to one of three possible memristance states: R_{on} (represented by "2" in the figure), R_{off} (represented by "0") and a memristance between R_{on} and R_{off} (represented by "1"). The use of multiple memristance states within the same array enables the implementation of applications such as pattern recognition.

The purpose of the simple example shown in Fig. 8 is to determine whether a 15-pixel image is a rectangle or a triangle. Even if the image is corrupted the circuit should be able to properly identify it most of the time. Assuming the image is represented by black and white pixels where a black pixel is represented by logic 1 and a white pixel is represented by logic 0, the circuit is mapped such that the output Y is 1 for a rectangle and 0 for a triangle. In the circuit, memristors connecting to pixels that are black only for rectangles are configured to R_{on} or 2. Likewise, memristors connecting to pixels that are black only for triangle images are set to R_{off} or 0 such that the output would tend toward logic 0 when a triangle is identified. Any pixel that is white for both images is connected to a "0" memristance and pixels that are black for both images are connected to a "1" memristance. Results (Table 2) show how the array of memristors shown in Fig. 8 can easily distinguish between the shapes shown in Fig. 9. The circuit was even

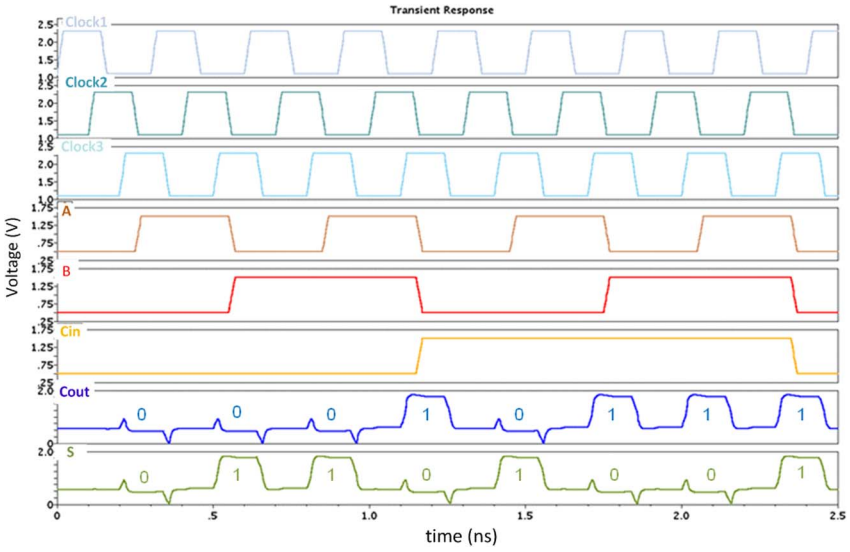

Fig. 7. *Simulation results for a PMLA mapped to implement a full adder as described in Fig. 6.*

able to identify "bad" versions of the shapes with imperfections present [27].

V. HYBRID CMOS/NANO THRESHOLD LOGIC GATES

While crossbar-array-based architectures are attractive due to the high device density achieved they also suffer from issues such as signal degradation and sneak path currents [16]. The PMLA provides a solution using NDR-based devices that can be used within a nanoscale fabric for signal restoration but these devices are rare and difficult to fabricate. Thus, another option that is worth exploring, especially for the near term, includes the use of CMOS circuitry at a fine granularity within the memristive fabric. In the circuits considered here, memristors are still used to implement the weights of the threshold logic as they are for the PMLA. However, CMOS is leveraged for the summation and comparison operations. Two options are explored here for implementing CMOS/nano threshold logic gates, one based on charge sharing and another based on current summation.

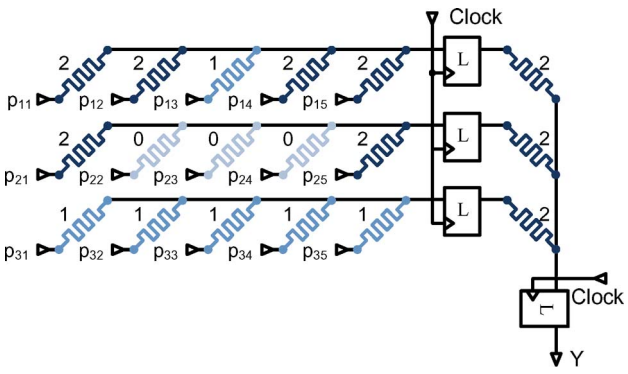

Fig. 8. *Configuration of a threshold logic array that can identify either a rectangle or a triangle [24].*

A. Charge Sharing Threshold Gates (CSTGs)

For the CSTGs, memristors are still used to implement the synapse where a voltage input is multiplied by the inverse of the memristance (the weight). However, a voltage divider between the memristor and a load resistor is used to produce a voltage signal representing the weighted input. This simple voltage divider circuit constitutes the synaptic functionality (i.e., weight) of the threshold gate and can be used to drive the gate of a transistor used for the summation and eventually comparison components of the circuit.

Fig. 10 illustrates the CSTG, including the synaptic voltage divider and CMOS circuitry used for summation and comparison/buffering. The CMOS circuit directly driven by the voltage divider must perform two major tasks: 1) amplify the voltage swing ΔV_n; and 2) provide an output that can be summed together with the outputs of other synapses. Summation in this circuit is accomplished through charge sharing at a common node (V_c) connecting all synaptic inputs. This implies that another function of the circuit directly driven by each synapse is to pass some amount of charge to the common node V_c that is directly proportional to the weighted sum.

The circuit functions by allowing a P-type metal–oxide–semiconductor logic (PMOS) transistor to charge up the node V_p between the two transistors when the input is 0 V. Once charged, this internal node is held high until

Table 2 Performance of Circuit in Fig. 8

Image	Y	Max. Power (μW)	Avg. Power (μW)	Delay
Good rectangle	1	1.66	0.57	40ps
Good triangle	0	2	0.61	40ps
Bad rectangle	1	2.06	0.66	40ps
Bad triangle	0	2.53	1.16	40ps

the input voltage is increased producing a nonzero voltage at V_n (based on the memristance value) which turns on the negative-channel metal–oxide–semiconductor (NMOS) transistor. A high input voltage (V_{in}) will also turn the PMOS off. The charge is then allowed to pass through the NMOS transistor from node V_p to internal node V_c. Since the voltage at node V_n is less than the threshold voltage of the NMOS transistor (high threshold variety) the transistor never turns on very strongly. Normally, the two floating capacitors at each internal node will charge to a common voltage. However, if the transistors are sized properly then the amount of charge and the associated voltage at the node V_c is weighted according to memristance values M_i and each of the inputs $V_{\text{in},i}$.

The next design parameter is V_{DD} which can be set to a lower voltage for 45-nm CMOS to help maximize ΔV_n relative to V_{DD} and force the circuit into subthreshold operation. For the examples considered here, V_{DD} is set to 250 mV. This leaves the sizing of the NMOS and PMOS transistors in the synapse circuit of Fig. 10. The size of the first PMOS (W_P) should be tuned to 1) react quickly to changes at the input V_{in}; and 2) provide enough capacitance at its drain to match the capacitance at V_c and promote charge sharing. Assuming the drain capacitance of a field-effect transistor (FET) is approximately equal to the gate capacitance, the sizing of W_P should be equal to the size of the first inverter driving the comparator/buffer circuits ($W_P = W_{pu} + W_{pd}$).

In the buffering side of the circuit, it is important that the first amplification stage be sensitive to changes at V_c. Thus, W_{pu} and W_{pd} should be sized such that the analog portion of the amplifier is centered on the average value of V_c. Since V_c was found to tend toward a relatively high voltage the amplifier/inverter should be skewed such that $W_{pu} \gg W_{pd}$. For this implementation, the proper ratio was found to be about 17 : 2 such that W_{pd} and W_{pu} are set to 210 nm and 1.79 μm, respectively. This also implies that the value of W_P ($\sim W_{pd} + W_{pu}$) should be 2 μm.

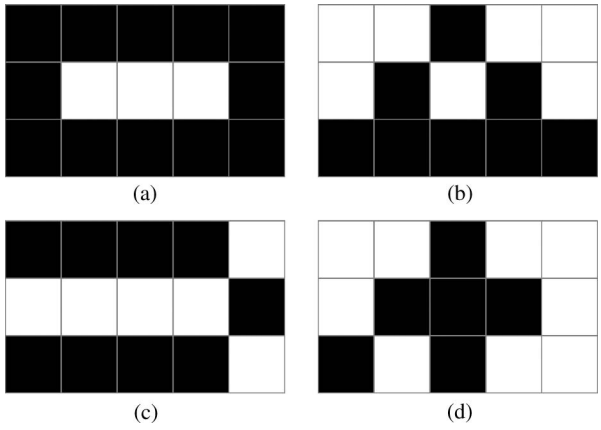

Fig. 9. *Analyzed images: (a) good rectangle; (b) good triangle; (c) bad rectangle; and (d) bad triangle.*

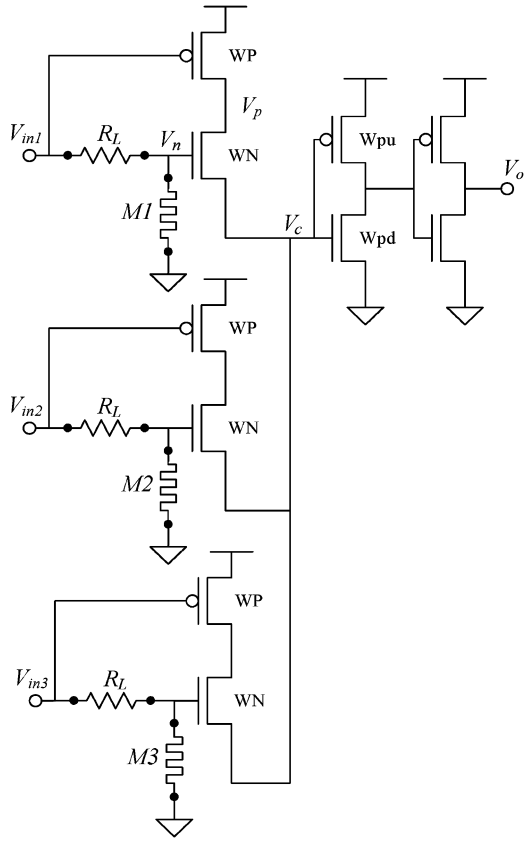

Fig. 10. *Subthreshold CMOS-memristor neural circuit consisting of the synaptic circuit and a buffering stage which amplifies the summed result of the synaptic circuits.*

Table 3 Energy and Delay for CSTGs Configured for Boolean Logic

Function	Memristance (MΩ)			Energy (10^{-15}J)	Transistor count	Delay (us)
	A	B	C			
ABC	31	31	31	48.9	13	103.2
A+B+C	72	72	72	46.48	13	279.4
A+BC	82	41	41	47.9	13	224.4
AB+AC+BC	47	47	47	48.27	13	168.3
AB+AC	57	28.5	28.5	49.03	13	257

Since the transistors are 45-nm CMOS and of the high V_{th} variety with V_{DD} set to 250 mV the energy consumption of the circuit is very low, on the order of tens or hundreds of femto-joules. However, the delay of the circuit is around 250 μs requiring very slow operation. Table 3 shows the performance of a three-input memristive CSTG circuit mapped to implement several Boolean logic functions. While such speeds may be on the order of what is observed for biological neurons, robust and even high performance operation will require massive parallelism.

As an example, simulation results are shown in Fig. 11 for three CSTGs configured to implement the majority logic full adder described earlier for the PMLA example. As can be seen from the figure the circuit as configured functions as desired. Furthermore, the average energy consumed for the complete full adder mapping is about 450 fJ. Most of this energy is due to static current through the memristors and could be reduced if 1) the delay is reduced; and 2) short pulses are used for the input signals. Such possibilities are to be considered as this circuit is further developed.

B. Current Mirror Threshold Gates (CMTGs)

The current mirror approach for implementing threshold gates is similar to the PMLA. Specifically, the weights of a threshold function are represented with memristance such that Ohm's Law is used to convert voltage signal inputs into a weighted input represented by a current. For a threshold gate using memristive devices for weights, if V_i and M_i are the voltage and memristance at the input i, then the output Y is

$$Y = \begin{cases} 0, & \text{if } \sum \dfrac{V_i}{M_i} < I_{\text{ref}} \\ 1, & \text{if } \sum \dfrac{V_i}{M_i} \geq I_{\text{ref}}. \end{cases} \tag{7}$$

The currents determined for each input i must be summed and then compared with a reference current I_{ref} representing the threshold. If the summed current exceeds the threshold (I_{ref}), the output of the gate is pulled to the high supply voltage or *vice versa*. This operation can be accomplished using current mirrors to first reflect the weighted inputs and sum them and then compare to the threshold represented by the reference I_{ref}.

A simple threshold circuit can be designed using a PMOS current mirror as a current comparator that can sum the currents flowing through the memristors and compare the summed current with I_{ref}. There are two important considerations for these current-mirror-based threshold gates. First, the current flowing through each memristor

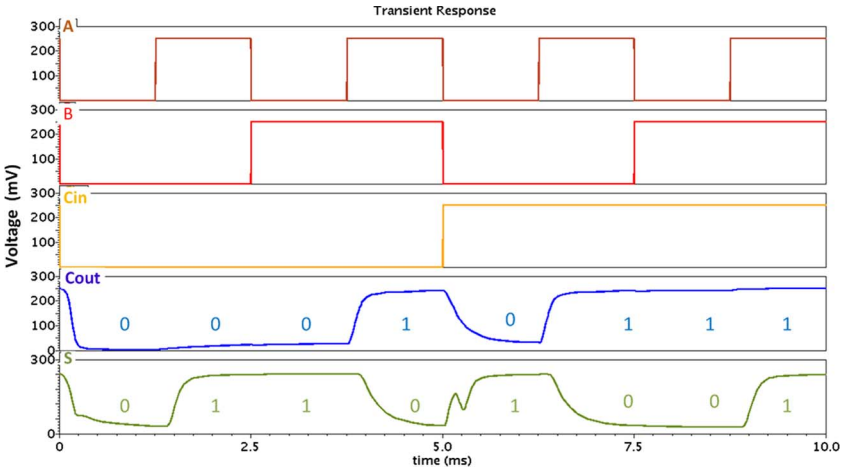

Fig. 11. *Simulation results showing the use of three three-input CSTGs in the construction of a full adder.*

should depend mainly on the corresponding memristance and input voltage. A naive design might connect one end of the memristor to the inputs and the other end to a common node to sum up the currents. However, in such a configuration, the current flowing through a memristor might not only depend on the memristance but also on the memristance values of memristors at other inputs. For example, if one of the inputs is grounded (logic 0) and its corresponding memristance is very low, then that input will serve as a current sink for all other inputs. The presence of this current sink which enables the reverse current flow from other inputs will reduce the expected summed current and so the threshold gate yields an erroneous output. To prevent this current flow in the reverse direction, a two-transistor NMOS current mirror on each input line is added. By adding the current mirror, current flowing through each input memristor primarily depends on the corresponding memristance and input voltage.

The second consideration is that the current comparator can be loaded by the inputs of the threshold gates at the next stage. To prevent this, a series of inverters are added which form an isolation circuit. The inverters not only isolate the current comparator from the external load but also "regulate" the output. The penultimate inverter in this series gives the functional output and the last inverter gives the inverted output of the function. These two inverters are scaled such that they can drive several gates. The drawback of these inverters is that they add to the delay of the circuit. A three-input threshold gate using memristors with the aforementioned additional circuitry is shown in Fig. 12. In general, an N-input threshold gate of this type requires $2N + 8$ transistors. For example, a three-input threshold gate requires 14 transistors.

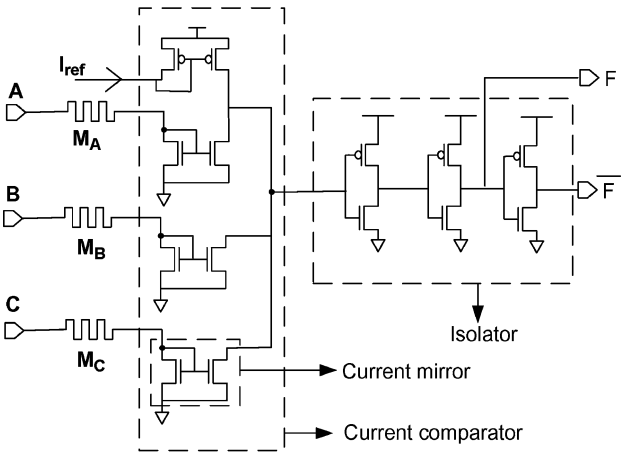

Fig. 12. *A three-input CMTG which uses the memristors as weights and I_{ref} as the threshold.*

Apart from the aforementioned circuit level concerns, there are some device level issues that can hamper the proper functioning of the CMTG. In the proposed circuit, there is an NMOS transistor in series with a memristor at each input. This implies that the current from a particular input is dependent not only on the memristance but also on the channel resistance of that NMOS transistor. To ensure that the current at the input depends primarily on the memristance (weight), the memristance should be greater than the NMOS channel resistance by at least two orders of magnitude. The lower limit for the memristive device modeled here is 121 kΩ, which is almost two orders of magnitude greater when compared to the channel resistance of the NMOS (45 nm) which is around 1–3 kΩ. Hence, the current flowing through the memristor depends mainly on the memristance and is almost independent of the NMOS channel resistance.

Continuing with the example of the full adder, Fig. 13 shows results for a CMTG-based logic circuit mapped to implement a full adder. As can be seen from the simulation, the circuit functions as expected and is actually faster than the CSTG circuit. Specifically, the subthreshold CSTG full adder operates with an average delay of 250 μs while the delay of the CMTG circuit is about 1 ns. This is important since the shorter data cycles also lead to less energy consumed per cycle, assuming the reference current is also relatively low.

VI. LOGIC SYNTHESIS FOR MEMRISTIVE THRESHOLD LOGIC

A bottom-up CAD framework is developed to map the weights to memristance values, synthesize a netlist using memristive threshold logic gates, and evaluate the performance from device level to system level using industry standard tools. It should be pointed out that the CMTG-style logic is used throughout this section for the CAD framework development due to the fact that CMTG exhibits a good balance between performance and reliability. That said, the same CAD algorithms described in this work can be easily adapted to any of the other circuit styles mentioned.

The CAD work begins with device modeling and circuit level performance analysis for the memristive threshold logic style (e.g., CMTG) using a transistor level simulation tool—Cadence Spectre in this case. The circuit level performance analysis is then used to characterize a library of threshold logic gates whereby the memristance values (weights) of each gate in the library are configured to implement particular Boolean logic functions. Characteristics and gate information from this library are then fed into Berkeley's SIS [20] for logic synthesis and the evaluation of system level performance.

Fig. 14 illustrates the design flow for synthesizing threshold-logic-based systems using the proposed

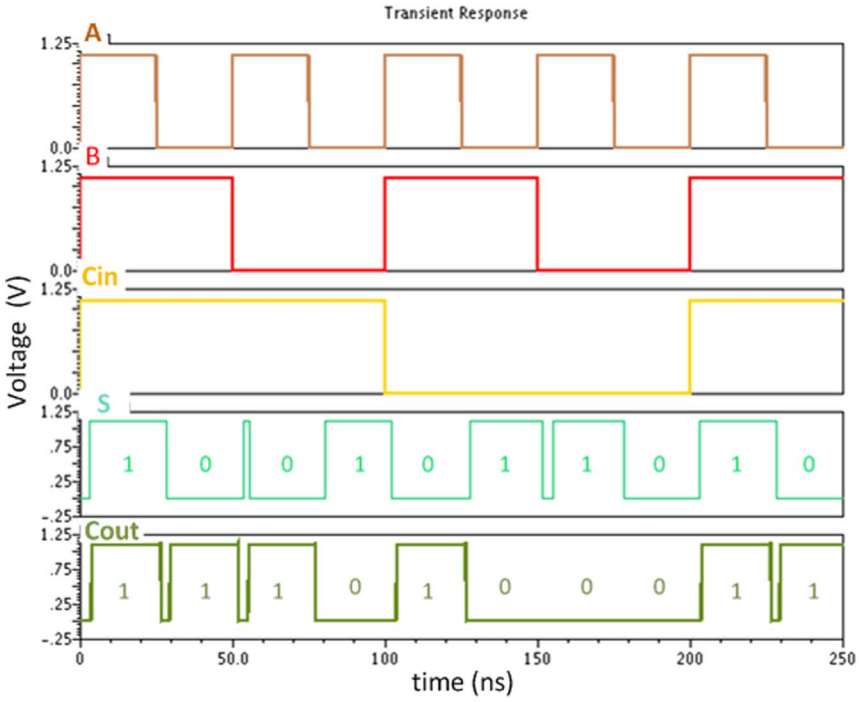

Fig. 13. ***Simulation results showing the use CMTGs in the construction of a full adder.***

memristive gates. As seen in the figure, the initial step is to accurately model a memristive device. This model is then used in the development of threshold logic gates such as those described earlier. Simultaneously, the weights of different Boolean functions are converted into memristance values using a mapping algorithm developed in house at the Polytechnic Institute of New York University (Brooklyn, NY) [31]. Performance results for various logic functions are obtained using circuit level simulations of threshold logic gates. A library of gates using these performance results is then created which is used by SIS to map the netlist. System level performance results are then obtained from SIS.

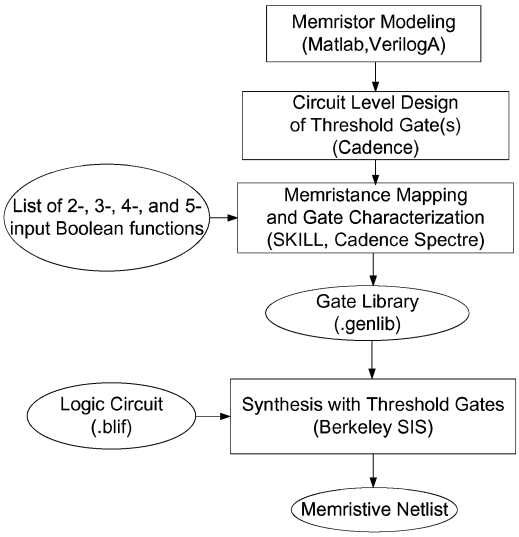

Fig. 14. ***A bottom-up CAD framework for memristive logic circuit implementation and performance evaluation. The boxes represent tools and the ovals represent output files.***

A. Performance Considerations for Memristance Mapping

In the threshold logic circuits described earlier, the memristor is directly interfaced with the gate or source/drain terminal of an NMOS transistor which causes an $RC(MC)$ delay. For CMTG, increasing the memristance will increase the RC delay and decreasing it will increase the power consumption of the circuit. Hence, an optimal memristance value should be chosen based on the power-delay tradeoff and this value is used to define the upper limit on the memristance that can be used as a weight to obtain optimal performance. This optimal memristance value can also be chosen as the average weight. For example, a simulation-based analysis to find the maximum permissible memristance based on the power consumption and delay through the CMTG circuit is shown in Fig. 15. It can be inferred from the figure that at low memristance, the delay is lower but power consumption is high while for the converse is true for a high memristance value. From this analysis using the CMTG circuit and the device model described earlier, the preferred memristance value to be

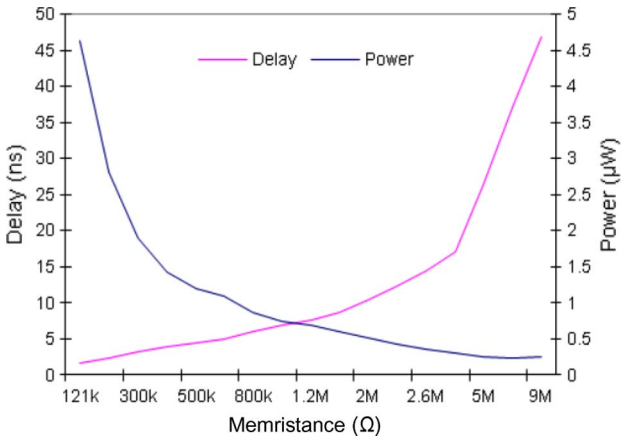

Fig. 15. *The power consumption and delay of the current comparator for different values of memristance. The memristance value 1.2 MΩ gives optimal power and delay performance.*

used for the weights is fixed at 1.2 MΩ for optimal delay and power performance. These trends are easily understood for CMTG since the memristor for this circuit style is directly connected to the source/drain terminal of the NMOS. It should be noted that for a circuit style such as CSTG the trend is different in that the delay will increase with increasing memristance since the memristive voltage divider of that circuit is connected to the gate of the transistor.

B. Logic Characterization via Memristance Mapping

While the threshold logic gate can implement any threshold logic function, in this research it was used to implement linearly separable Boolean functions and their complements. The minimum weights and the corresponding threshold for a threshold gate to implement all three-input, four-input, and five-input linearly separable Boolean functions are listed in [28]. A threshold gate can be programmed to implement a Boolean function by changing the weights, the threshold, or both. For example, a two-input threshold gate with threshold "2," implements an AND gate if the weights at its inputs are both "1," and implements an OR gate if the weights are changed to "2." Different Boolean functions may have their weights described for different thresholds. If the threshold is fixed then these weights have to be recalculated for the common threshold value such that we can implement all of the functions by simply adjusting the weights (memristance) and keeping the threshold value fixed. For the gates considered here, different Boolean functions are implemented by adjusting the memristance (weights) while the reference signal (threshold) is fixed.

Algorithm 1 is used to convert the weights for different functions which have different thresholds to weights for a common threshold and to map these weights to associated memristance values [31].

Algorithm 1: Adjusting weights and calculating the memristances for a fixed threshold

1: $B \leftarrow$ number of functions that can be implemented
2: $T_i \leftarrow$ threshold of the ith Boolean function
3: $W_{i,j} \leftarrow$ weight at jth input of ith function
4: $M_{i,j} \leftarrow$ memristance at jth input of ith function
5: $M_{\max} \leftarrow$ maximum memristance possible
{**Calculate common threshold**: T_c}
6: $T_c =$ LCM of T_i, T_c $\forall i \in B$
{**Adjusted weights**}
7: $W^r_{i,j} = W_{i,j}(T_c/T_i)$ $\forall i \in B$
8: $W^r_{\min,N} = \text{Minimum}(W^r_{i,j})$ $\forall i : 1$ to $B, j : 1$ to N
9: $W^r_{\min,N} = M_{\max}$
{**Calculate memristance**}
10: $\text{Mem}_{i,j} = M_{\max}(W^r_{\min,N}/W^r_{i,j})$ $\forall i$:1 to B, j:1 to N

For example, consider two Boolean functions ABC and $A + BC$. For the function ABC, the weights at the inputs are all (1, 1, 1) and the threshold is 3 while for the function $A + BC$ the weights are (2, 1, 1) and the threshold is 2. The common threshold is determined by taking the least common multiple (LCM) of the threshold for the different Boolean functions using step 6 in the algorithm. For this example, the common threshold for ABC and $A + BC$ is LCM $(3, 2) = 6$. The common threshold varies when the number of inputs varies due to changes in the LCM. The weights for all of the functions are then obtained using steps 7–9 by adjusting the original weights for the common threshold. In the example, the adjusted weights at the inputs are (2, 2, 2) for the function ABC and (6, 3, 3) for the function $A + BC$. These adjusted weights must then be converted into corresponding memristance values using step 10. For the CMTG circuit, the maximum memristance $M_{\max}$ is fixed at 1.2 MΩ based on optimal power-delay analysis. In the example, the memristance values are (1200, 1200, and 1200 kΩ) for the function ABC and (400, 800, and 800 kΩ) for the function $A + BC$. For further analysis, adjusted weights and the respective memristance values are calculated for two-, three-, four-, and five-input Boolean functions. Table 4 shows the values of all three-input Boolean functions calculated using Algorithm 1.

C. Characterizing the Library of Mapped Gates

Once the memristance values required to implement the Boolean functions are calculated, energy consumption and delay are determined for all gates mapped to particular Boolean functions by configuring their corresponding memristors. Furthermore, a library of these functions along with their performance is created for logic synthesis. The memristors are configured to a memristance value by setting the φ parameter in equation (4) which corresponds to the history of the voltage applied across the memristor. The Cadence scripting language SKILL is then used to configure the memristors for exhibiting different Boolean

Table 4 Weights and Adjusted Weights for Three-Input Boolean Functions Calculated With Algorithm 1

Threshold	3	1	2	2	Common threshold = 6			
	Weights				Adjusted weights for common threshold			
Functions	ABC	A+B+C	AB+BC+CA	A+BC	ABC	A+B+C	AB+BC+CA	A+BC
Input A	1	1	1	2	2	6	3	6
Input B	1	1	1	1	2	6	3	3
Input C	1	1	1	1	2	6	3	3

functions, run the simulations, and obtain the respective energy and delay values for each Boolean function/gate. The energy consumption, transistor count, and delay for some of the Boolean functions mapped to CMTG circuits are listed in Table 5. For these circuits, the energy consumption varies with the number of inputs due to a change in the reference current (the common threshold). Since a gate implements a function and its complement, the total energy consumption includes the energy consumed while exhibiting a Boolean function and its complement. For the same number of inputs, the energy consumed for implementing different functions is more or less the same. However, for CMTGs, as the number of inputs is increased, the energy consumption increases due to an increase in the reference current. For example, consider the energy consumption for a two-input AND gate (9.1×10^{-15} J), whose reference current is 0.8 μA and a three-input AND gate (6.8×10^{-15} J), whose reference current is 2 μA. Since the reference current for the two-input gate is low it has lower energy consumption when compared to the three-input gate which has a higher reference current.

The transistor count includes the number of transistors required to implement a function and its complement. The transistor count increases linearly (by two transistors for each input) when compared to the exponential increase for pure CMOS LUTs used to implement most FPGAs. As mentioned previously, the transistor count for an N-input CMTG gate is given by $(2N + 8)$ transistors. Every additional input needs an extra memristor and a current mirror (two NMOS transistors), while the PMOS current mirror and the buffering circuit remain the same. Since the size of the memristor is very small (about 50 nm $\times$ 50 nm $\times$ 50 nm for TiO_2 memristors), the number of memristors required is not taken into account.

The delay of a gate depends on the type of function that is implemented. For example, for the same number of inputs, the delay of an OR gate is greater when compared to that of an AND gate. This is because the memristance at the input of an OR gate is smaller than that of an AND gate. The memristance values for the function AB are (1.2 and 1.2 MΩ) while for function $A + B$ the memristance values are (600 and 600 kΩ). The AND gate with higher memristance has higher delay (1.97 ns) while the OR gate exhibits lower delay (0.982 ns).

The energy and delay information for two-, three-, four-, and five-input Boolean functions that can be implemented using threshold logic were obtained from Cadence Spectre simulations. A SKILL script was written to create a library of mapped gates which contains the energy and delay information of these gates and SIS uses this library for logic synthesis. This library is similar to .genlib format [29] with the only difference being the area field is replaced with energy consumption data.

To evaluate the performance of the proposed gates at the system level, logic level benchmarking netlists are synthesized using the library of gates. This logic synthesis is done using the Berkeley SIS 2.0 [29] logic synthesis tool. For demonstration purposes, netlists from the ISCAS-85 combinational benchmarks in .blif format are synthesized for the developed library of gates using this tool. The SIS tool also provides the energy consumption, critical path delay and gate count using the performance metrics obtained from Cadence Spectre for characterizing the library.

Fig. 16 shows the energy-delay product (EDP) results for a four-input CMOS LUT-based FPGA, a CMOS-capacitive threshold logic (CTL) circuit [19], the memristive CMTG implementation, and the PMLA. Not shown in Fig. 16 are results for CSTG which is worst in terms of

Table 5 Characteristics of Some CMTGs Mapped to Boolean Logic

Function	Memristance (MΩ)			Energy (10^{-15}J)	Transistor count	Delay (ns)
	A	B	C			
AB	1.2	1.2		9.10	12	1.97
A+B	0.6	0.6		8.98	12	0.982
ABC	1.2	1.2	1.2	6.8	14	1.92
A+B+C	0.4	0.4	0.4	7.29	14	0.637
AB+BC+CA	0.8	0.8	0.8	6.65	14	1.19
AB+AC	0.6	1.2	1.2	6.93	14	0.928

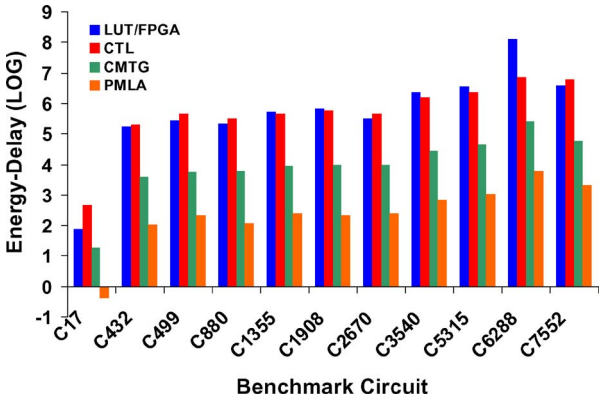

Fig. 16. *EDP for the ISCAS-85 benchmarks implemented using four-input LUT, CTL, CMTG, and PMLA circuits.*

energy delay since the delay in the current implementation is so high. However, CSTG is still promising in terms of power and it may also be possible that other memristor technologies could help reduce the delay.

From Fig. 16, it can also be seen that the CMTG circuit shows less EDP than the two pure CMOS implementations (LUT and CTL) by at least an order of magnitude, while the CTL and LUT circuits exhibit the same order of magnitude of EDP relative to one another. The most efficient circuit according to Fig. 16 is the PMLA which includes very little CMOS and is mainly built from memristors and NDR devices. Of course, the drawback of the PMLA itself may be in the lack of availability of NDR devices that can be easily integrated with memristors and/or CMOS. The benchmark circuit C6288 is a 16 × 16 multiplier which uses a few hundred full adders. Hence, the EDP is very high for all implementations of this particular benchmark.

VII. CONCLUSION AND PERSPECTIVES FOR THE FUTURE

This work showcases the versatility and potential of the memristor when applied in several nano and hybrid CMOS/nano logic implementations. Different flavors of logic circuits varying from Boolean to threshold logic were designed and realized and comparisons between the proposed circuits were shown. A design flow that utilizes industry standard tools and an inhouse CAD framework was developed for the logic synthesis of the proposed memristive threshold logic.

A total of five memristive circuits were presented which can be used to implement logic at the nanoscale. The crossbar-array-based circuits (PLA, PMLA, and threshold logic array) are promising for their high-density and low-power operation. However, the purely nanoscale-array-based circuits considered either suffer from signal degradation or require rare NDR-based devices. A hybrid CMOS/nano circuit based on charge sharing (CSTG) was also presented which can operation in subthreshold mode for ultralow power consumption. However, the low-power operation of this circuit comes at the expense of high delay. It is possible that different design parameters (e.g., memristance or power supply) could be tailored to improve the speed of CSTG logic but further research is required. Finally, a second hybrid CMOS/nano circuit based on current summation (CMTG) was presented that also exhibits low power consumption but with less delay penalty.

Examples of the proposed CMTG and PMLA circuits were synthesized using the aforementioned design flow and comparisons were made to CMOS FPGA (LUT) and threshold (CTL) circuits with respect to energy delay for ISCAS-85 benchmarks. It was shown that the proposed nano and hybrid CMOS/nano circuits trumped the existing CMOS circuits with lower energy-delay metrics with the PMLA showing the best results. The PMLA's efficiency can be attributed to its predominantly nano implementation and with further progress in the development of NDR devices, this circuit holds great promise. However, for the near future the hybrid CMOS/nano CMTG circuit is preferred for its CMOS compatible fabrication and low energy delay that beats pure CMOS circuits by an order of magnitude.

Another critical challenge is the difficulty in interfacing nanoscale memristors with CMOS technology. Although our models (memristors, vias, etc.) accurately represent the physical attributes of the circuit elements, further work is needed to develop an optimized framework for the physical hybrid CMOS/nano integrated circuit design. With ever increasing advancements in the CMOS integrated circuit (IC) fabrication technology, our proposed circuits could very well be realized in the near future. Several researchers such as [30] have already worked on designing and fabricating hybrid CMOS/nano field-programmable architectures.

Moving forward, methodologies that provide for a fast write circuitry will be developed and implemented for the proposed circuits. The write circuitry will aim to leverage the exponential drift of memristors as mentioned in Section II to provide for write times in the order of tens of microseconds as opposed to write times in the order of hundreds of milliseconds for memristors programmed for linear drift. Another application of the write circuitry will be to extend the proposed threshold circuits to implement neuromorphic applications by allowing the gates to learn.

Another feature of interest is the development of variation-aware algorithms to configure the fabrics considered. The impact of variations in memristors is significant and warrants a methodology that can provide optimum memristances (weights) based on the tradeoff between variation tolerance and performance requirements. The write/learn circuitry can also be designed to provide for variations tolerance.

In short, the memristive logic circuits presented have been shown to be energy efficient and in some cases may be able to achieve processing speeds comparable to that of CMOS. This work takes an important step towards developing a clear understanding of how the property of memristance can be harnessed to develop robust nanoscale computing systems. ■

Acknowledgment

The authors would like to thank H. (Helen) Li and O. Sinanoglu, both of New York University, for many interesting discussions on the topics presented here. They would also like to thank N. McDonald and Q. Wu of Air Force Research Labs for discussions related to the memristor device models used to make this work possible.

REFERENCES

[1] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, pp. 888–900, 2005.

[2] A. DeHon, "Array-based architecture for FET-based, nanoscale electronics," *IEEE Trans. Nanotechnol.*, vol. 2, no. 1, pp. 23–32, Mar. 2003.

[3] S. Kim, Y. Zhang, B. Lee, M. Caldwell, and H.-S. P. Wong, "Fabrication and characterization of emerging nanoscale memory," in *Proc. IEEE Symp. Circuits Syst.*, 2009, pp. 65–68.

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

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

[6] Y. N. Joglekar and S. J. Wolf, "The elusive memristor: Properties of basic electrical circuits," *Eur. J. Phys.*, vol. 30, no. 4, pp. 661–675, 2009.

[7] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, Mar. 2010.

[8] G. Snider, "Spike-timing-dependent learning in memristive nanodevices," in *Proc. Memristor Memristive Syst. Symp.*, 2008, pp. 85–92.

[9] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *Proc. Int. Workshop Cellular Nanoscale Netw. Appl.*, 2010, DOI: 10.1109/CNNA.2010.5430304.

[10] Y. Ho, G. M. Huang, and P. Li, "Nonvolatile memristor memory: Device characteristics and design implications," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 485–490.

[11] H. Kim, M. Sah, C. Yang, and L. Chua, "Memristor-based multilevel memory," in *Proc. Int. Workshop Cellular Nanoscale Netw. Appl.*, 2010, DOI: 10.1109/CNNA.2010.5430320.

[12] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams, "Memristor-CMOS hybrid integrated circuits for reconfigurable logic," *Nano Lett.*, pp. 3640–3645, Sep. 2009.

[13] W. Wang, T. T. Jing, and B. Butcher, "FPGA based on integration of memristors and CMOS devices," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1963–1966.

[14] L. O. Chua. (2008). Memristors. *Proc. Memristor Memristive Syst. Symp.* [Online]. Available: http://www.youtube.com/watch?v=QFdDPzcZwbs

[15] D. B. Strukov and R. S. Williams, "Exponential ionic drift: Fast switching and low volatility of thin-film memristors," *Appl. Phys. A*, vol. 94, no. 3, pp. 515–519, 2008.

[16] M. M. Ziegler and M. R. Stan, "Design and analysis of crossbar circuits for molecular nanoelectronics," in *Proc. IEEE Conf. Nanotechnol.*, 2002, pp. 323–327.

[17] C. P. Collier, J. O. Jeppesen, Y. Luo, J. Perkins, E. W. Wong, J. R. Heath, and J. F. Stoddart, "Molecular-based electronically switchable tunnel junction devices," *J. Amer. Chem. Soc.*, vol. 123, no. 50, pp. 12 632–12 641, Dec. 2001.

[18] G. S. Rose and M. R. Stan, Jr., "A programmable majority logic array using molecular scale electronics," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2380–2390, Nov. 2007.

[19] V. Beiu, J. Quintana, and M. Avedillo, "VLSI implementations of threshold logic-a comprehensive survey," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1217–1243, Sep. 2003.

[20] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Synthesis and optimization of threshold logic networks with application to nanotechnologies," in *Proc. Conf. Design Autom. Test Eur.*, 2004, pp. 904–909.

[21] E. Goto, K. Murata, K. Nakazawa, K. Nakagawa, T. Moto-Oka, Y. Ishibashi, T. Soma, and E. Wada, "Esaki diode high-speed logical circuits," *IRE Trans. Electron. Comput.*, vol. EC-9, no. 1, pp. 25–29, Mar. 1960.

[22] G. S. Rose, A. C. Cabe, N. Gergel-Hackett, N. Majumdar, M. R. Stan, J. C. Bean, L. R. Harriott, Y. Yao, and J. M. Tour, "Design approaches for hybrid cmos/molecular memory based on experimental data," in *Proc. Great Lakes Symp. Very Large Scale Integr. (VLSI)*, May 2006, pp. 2–7.

[23] D. Goldhaber-Gordon, M. S. Montemerlo, J. C. Love, G. J. Opiteck, and J. C. Ellenbogen, "Overview of nanoelectronic devices," *Proc. IEEE*, vol. 85, no. 4, pp. 521–540, Apr. 1997.

[24] S. C. Goldstein and M. Budiu, "Nanofabrics: Spatial computing using molecular nanoelectronics," in *Proc. 28th Annu. Int. Symp. Comput. Archit.*, Jun. 2001, pp. 178–189.

[25] W. Wang, K. Walus, and G. A. Jullien, "Quantum-dot cellular automata adders," in *Proc. IEEE Conf. Nanotechnol.*, Aug. 2003, pp. 461–464.

[26] M. M. Ziegler and M. R. Stan, "CMOS/nano co-design for crossbar-based molecular electronic systems," *IEEE Trans. Nanotechnol.*, vol. 2, no. 4, pp. 217–230, Dec. 2003.

[27] J. Rajendran, H. Manem, and G. Rose, "NDR based threshold logic fabric with memristive synapses," in *Proc. IEEE Conf. Nanotechnol.*, Jul. 2009, pp. 725–728.

[28] S. Muroga, *Threshold Logic and Its Applications*. New York: Wiley, 1971.

[29] Berkeley SIS. [Online]. Available: http://www-cad.eecs.berkeley.edu/_pchong/sis.html

[30] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, no. 3, 2007, DOI: 10.1088/0957-4484/18/3/035204.

[31] J. Rajendran, H. Manem, R. Karri, and G. Rose, "An energy-efficient memristive threshold logic circuit," *IEEE Trans. Comput.*, 2011.

[32] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1985, pp. 695–698.

ABOUT THE AUTHORS

Garrett S. Rose (Member, IEEE) received the B.S. degree in computer engineering from Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from the University of Virginia, Charlottesville, in 2003 and 2006, respectively. His Ph.D. dissertation was on the topic of circuit design methodologies for molecular electronic circuits and computing architectures.

Currently, he is with the Air Force Research Laboratory, Information Directorate, Rome, NY where his work is focused on nanoelectronic computing research. Prior to that, from August 2006 to May 2011, he was an Assistant Professor in the Department of Electrical and Computer Engineering at the Polytechnic Institute of New York University, Brooklyn. From May 2004 to August 2005, he was with the MITRE Corporation, McLean, VA, involved in the design and simulation of nanoscale circuits and systems. His research interests include low-power circuits, system-on-chip design, 3-D integrated circuits, and developing very large scale integration (VLSI) design methodologies for novel nanoelectronic technologies.

Dr. Rose is a member of the Association of Computing Machinery, IEEE Circuits and Systems Society, and IEEE Computer Society. He serves and has served on Technical Program Committees for several IEEE conferences (including ISCAS, GLSVLSI, NANOARCH) and workshops in the area of VLSI design. In 2010, he was a guest editor for a special issue of the *ACM Journal of Emerging Technologies in Computing Systems* that presents key papers from the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'09).

Jeyavijayan Rajendran (Student Member, IEEE) received the B.E. degree in electronics and communication engineering from Anna University, Chennai, Tamil Nadu, India, in 2008 and the M.S. degree in computer engineering from the Polytechnic Institute of New York University, Brooklyn, in 2010, where he is currently working towards the Ph.D. degree in the Electrical and Computer Engineering Department.

His research interests include nanoscale architectures and hardware trust.

Harika Manem (Student Member, IEEE) received the M.S. degree in electrical engineering from the Polytechnic Institute of New York University, Brooklyn, in 2008, where she is currently working towards the Ph.D. degree in the Department of Electrical and Computer Engineering.

Her research primarily focuses on investigating novel and innovative emerging technologies and applying them in the realization of fast, highly compact, and power conservative nanoelectronic systems that could eventually replace existing systems.

Ramesh Karri (Member, IEEE) received the B.E. degree in electronics and communications engineering from Andhra University, Visakhapatnam, India, in 1985, the M.S. degree in computer science from University of Hyderabad, Hyderabad, India, in 1988, and the M.S. degree in computer engineering and the Ph.D. degree in computer science from the University of California San Diego, La Jolla, in 1992 and 1993, respectively.

He is a Professor in the Electrical and Computer Engineering Department, Polytechnic Institute of New York University, Brooklyn. His research interests include trustworthy hardware design, the interaction between security and reliability, and nanoscale architectures. He has published over a 100 conference and journal articles in these areas. In 2011, a paper he has coauthored on memristor-based logic design has received the best student paper award at the IEEE VLSI Design Conference. In 2009, a Ph.D. dissertation that he has advised entitled "Secure hardware design and test" was awarded the 3rd prize in the IEEE Test Technology Technical Council dissertation contest. In 2003, a Ph.D. dissertation he has advised "On dependable and secure hardware design," received the outstanding award in the new directions in circuit and system test from the European Design and Automation Association. He has also received the NSF CAREER Award and the Alexander Humboldt Fellowship.

Dr. Karri is a member of the IEEE Computer Society. He is the founding chair of the IEEE Technical Committee on Nanoelectronics, Nanoarchitectures and Nanocomputing (TC NANO). He co-founded IEEE Symposium Nanoscale Architectures (NanoArch) and has served as its General Co-Chair during its formative years. He is serving as the program chair of the 2012 IEEE Symposium on Hardware oriented Security and Trust (HOST 2012). He is an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION FORENSICS and the ACM *Journal of Emerging Technologies in Computing Systems*. He has served or is currently serving on several conference program committees.

Robinson E. Pino (Senior Member, IEEE) received the B.E. degree in electrical engineering with honors (*summa cum laude*) from the City University of New York, City College, in 2002 and the M.Sc. and Ph.D. degrees in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2003 and 2005, respectively.

Since 2009, he has been a Senior Electronics Engineer at the United States Air Force Research Laboratory (AFRL), Rome, NY, where he is a Program Manager and Principle Scientist for the Computational Intelligence and Neuromorphic Computing research efforts. His expertise is within technology development and program management. His professional experience expands government, industry, and academia. Within Industry, he worked from 2005 to 2009 at IBM as an Advisory Scientist/Engineer Development enabling advanced CMOS technologies, and as a Business Analyst within IBM's Photomask business unit. In this later capacity, he was responsible for development and manufacturing spending, capacity planning, lean manufacturing, and business process automation. Within academia, he served during 2006–2009 as an adjunct professor at the University of Vermont where he taught electrical engineering courses at the graduate and undergraduate levels. He holds two patents, four pending, and has published over 40 technical papers including one book. Some of his technical accomplishments include the measurement and characterization for the first time of the Burstein-Moss Shift effect in ternary III-V compound semiconductors and developed an encapsulation methodology for the adhesion free growth of AlSb bulk crystals in silica crucibles.

Dr. Pino was named Distinguished Lecturer of IEEE, EDS, in 2010, AFRL Information Directorate Scientist/Engineer of the year in 2011, IEEE-Mohawk Valley Section Engineer of the Year in 2011, National Science Foundation IGERT Fellow in 2004, and named Top 200 Most Influential Hispanics in Technology by *HE&IT* Magazine in 2011. Upon graduation from the Ph.D. program, he received the Allen B. DuMont Prize that is awarded to a graduate student who has demonstrated high scholastic ability and has made a substantial contribution to his/her field of study.