This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies

Shahar Kvatinsky, *Student Member, IEEE*, Guy Satat, Nimrod Wald, Eby G. Friedman, *Fellow, IEEE*, Avinoam Kolodny, *Senior Member, IEEE*, and Uri C. Weiser, *Fellow, IEEE*

***Abstract*— Memristors are novel devices, useful as memory at all hierarchies. These devices can also behave as logic circuits. In this paper, the IMPLY logic gate, a memristor-based logic circuit, is described. In this memristive logic family, each memristor is used as an input, output, computational logic element, and latch in different stages of the computing process. The logical state is determined by the resistance of the memristor. This logic family can be integrated within a memristor-based crossbar, commonly used for memory. In this paper, a methodology for designing this logic family is proposed. The design methodology is based on a general design flow, suitable for all deterministic memristive logic families, and includes some additional design constraints to support the IMPLY logic family. An IMPLY 8-bit full adder based on this design methodology is presented as a case study.**

***Index Terms*— Design methodology, IMPLY, logic, memristive systems, memristor, Von Neumann architecture.**

I. Introduction

MEMRISTORS [1] and memristive devices [2] are novel structures, useful in many applications. These devices are basically resistors with varying resistance, which depends on the history of the device. It can be used for memory, where the data is stored as a resistance. While memory is the common application for memristive devices, additional applications can also use memristive devices as functional blocks, such as analog circuits, neuromorphic systems, and logic circuits. Although the definition of memristive devices is broader than the definition of memristors, it is common to use the term memristor for all memristive devices [10], [11]. In this paper, for simplicity, the terms memristor and memristive device are used interchangeably.

The use of memristors to perform logical operations has been proposed in several different ways. In some logic families, memristors are integrated with CMOS structures to perform the logical operation, while the logical values are represented by voltage levels. In [3], memristors are used as a reconfigurable switch. In [4], a hybrid memristor-CMOS logic family is proposed, memristor ratioed logic (MRL). In MRL, the memristors act as computational elements, performing OR and AND Boolean functions, while the CMOS transistors perform logical inversion and amplification of the logical voltage signals. A similar approach is proposed in [5].

Another approach for logic with memristors is to treat resistance as the logical state, where the high and low resistance are considered, respectively, as logical zero and one. For this approach, the memristors are the primary building blocks of the logic gate. Each memristor acts as an input, output, computational logic element, and latch in different stages of the computing process [6]. This approach is suitable for crossbar array architectures and can therefore be integrated within a standard memristor-based crossbar, commonly used for memory. This approach is appealing since it provides an opportunity to explore advanced computer architectures different from the classical von Neumann architecture. In these architectures, the memory can perform logical operations on the same devices that store data, i.e., performing computation inside the memory. This paper focuses on this approach.

Material implication (IMPLY logic gate) [7] is one example of a basic logical element using this approach, combining state memory and a Boolean operator. Additional logic families, which extends the IMPLY logic gate by using certain variations of a regular memristor-based crossbar, have also been proposed [8], [9] and are not considered in this paper. A specific modification of the crossbar structure is, however, presented in this paper to enhance the performance of the logic gate.

In this paper, the IMPLY logic gate is described in Section III, and a memristor-based crossbar in Section IV. A design methodology for the IMPLY logic gate is proposed in Section V. This design methodology consists of a design flow appropriate for all memristor-based logic families, as well as the IMPLY logic family. This design methodology is demonstrated by a case study of an 8-bit IMPLY full adder in Section VI. Logic inside a memristor-based memory is discussed in Section VII. This paper is concluded in Section VIII.

II. Memristors

Memristors were conceived in 1971 by Chua [1] based on fundamental principles of symmetry. Chua proposed a fourth fundamental electronic component in addition to the three already well-known fundamental electronic components: the resistor, capacitor, and inductor. The memristor has varying

Manuscript received February 23, 2013; revised June 1, 2013 and August 26, 2013; accepted September 8, 2013. This work was supported in part by the Hasso Plattner Institute, in part by the Advanced Circuit Research Center at Technion, and in part by the Intel Collaborative Research Institute for Computational Intelligence.

S. Kvatinsky, G. Satat, N. Wald, A. Kolodny, and U. C. Weiser are with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel (e-mail: skva@tx.technion.ac.il; guysatat@hotmail.com; nimrodwald@gmail.com; kolodny@ee.technion.ac.il).

E. G. Friedman is with the Department of Electrical Engineering and Computer Engineering, University of Rochester, Rochester, NY 14627 USA (e-mail: friedman@ece.rochester.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/TVLSI.2013.2282132

Fig. 1. Memristive device symbol. The thick black line on the left side of the device represents the polarity of the device. If the current flows into the device, the resistance of the device decreases. If the current flows out of the device, the resistance increases.

resistance (also named memristance). Changes in the memristance depend upon the history of the device (e.g., the memristance may depend on the total charge passing through the device, or alternatively, on the integral over time of the applied voltage across the ports of the device).

The theory of memristors was extended to memristive devices in 1976 [2]. Formally, a current-controlled time-invariant memristive system is represented by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, i) \tag{1}$$

$$v(t) = R(x, i) \cdot i(t) \tag{2}$$

where x is an internal state variable, i(t) is the memristive device current, $v(t)$ is the voltage of the memristive device, $R(x, i)$ is the memristance, and t is time. The symbol of a memristor is illustrated in Fig. 1. Note that the polarity of the symbol defines the sign (positive or negative) of the current.

Since Hewlett–Packard announced the fabrication of a working memristor in 2008 [12], there has been increasing interest in memristors and memristive systems. New devices exhibiting memristive behavior have been announced [13], [14], and existing devices such as spin-transfer torque magnetoresistive random access memory (STT-MRAM) have been redescribed in terms of memristive systems [15]. Actually, most emerging memory technologies obey (1) and (2) and can therefore be described as memristive devices or memristors [11].

Several memristor models have been proposed to describe the behavior of physical memristors [16]–[23]. These models are deterministic and do not consider stochastic switching [40], [41]. In this paper, the threshold adaptive memristor (TEAM) model [23] is used. In the TEAM model, memristors have an adaptive nonlinearity and a current threshold. For this model, (1) becomes

$$\frac{dx(t)}{dt} = \begin{cases} k_{\mathrm{OFF}} \cdot \left(\frac{i(t)}{i_{\mathrm{OFF}}} - 1\right)^{\alpha_{\mathrm{OFF}}} \cdot f_{\mathrm{OFF}}(x), & 0 < i_{\mathrm{OFF}} < i \quad \text{3(a)} \\ 0, & i_{\mathrm{ON}} < i < i_{\mathrm{OFF}} \quad \text{3(b)} \\ k_{\mathrm{ON}} \cdot \left(\frac{i(t)}{i_{\mathrm{ON}}} - 1\right)^{\alpha_{\mathrm{ON}}} \cdot f_{\mathrm{ON}}(x), & i < i_{\mathrm{ON}} < 0 \quad \text{3(c)} \end{cases}$$

where k_{OFF} and k_{ON} are fitting parameters, α_{ON} and α_{OFF} are the adaptive nonlinearity parameters, i_{OFF} and i_{ON} are the current threshold parameters, and $f_{\mathrm{ON}}(x)$ and $f_{\mathrm{OFF}}(x)$ are window functions. An I–V curve for the TEAM model is shown in Fig. 2 for memristors where (2) is

$$v(t) = \left[R_{\mathrm{ON}} + \frac{R_{\mathrm{OFF}} - R_{\mathrm{ON}}}{x_{\mathrm{OFF}} - x_{\mathrm{ON}}}(x - x_{\mathrm{ON}})\right] \cdot i(t) \tag{4}$$

where R_{ON} and R_{OFF} are, respectively, the minimum and maximum resistance of the memristor, and x_{ON} and x_{OFF} are, respectively, the minimum and maximum allowed value of the internal state variable x.

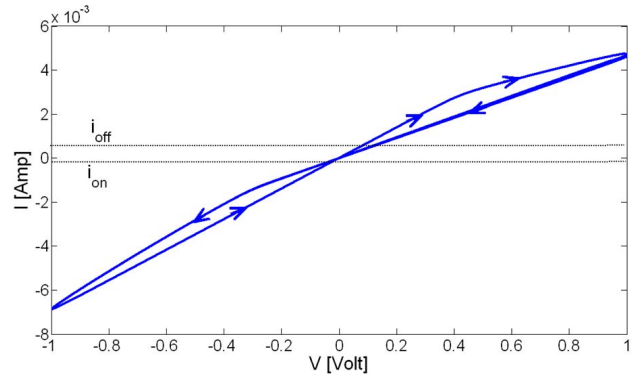


Fig. 2. I–V curve of a memristor based on the TEAM model driven with a sinusoidal input of 1 volt, where $R_{\mathrm{ON}} = 50\Omega$, $R_{\mathrm{OFF}} = 1$ kΩ, $k_{\mathrm{OFF}} = 1.46e^{-9}$ nm/s, $\alpha_{\mathrm{OFF}} = 10$, $i_{\mathrm{OFF}} = 115\ \mu$A, $k_{\mathrm{ON}} = -4.68e^{-13}$ nm/s, $\alpha_{\mathrm{ON}} = 10$, $i_{\mathrm{ON}} = 8.9\ \mu A$, $x_{\mathrm{ON}} = 1.2$ nm, and $x_{\mathrm{OFF}} = 1.8$ nm.

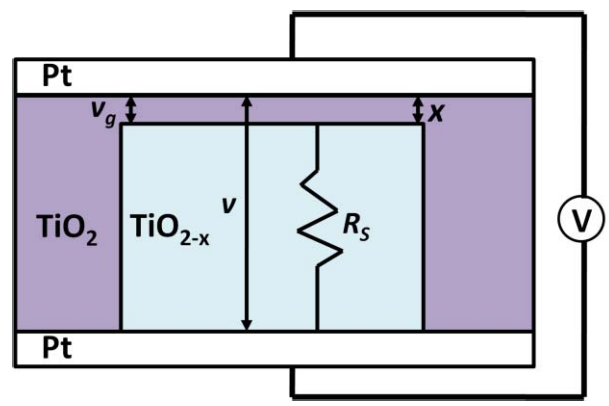


Fig. 3. Schematic of the physical model proposed in [20] for a TiO_2 memristor.

Memristors are nonvolatile and compatible with standard CMOS technologies [24]. These devices are fabricated in the metal layers of an integrated circuit, where the memristive effects occur in the oxide between the metal layers (e.g., in TiO_2 and TaO_x) [25] or within the metal layers (e.g., in STT-MRAM). The physical model of a TiO_2 memristor, proposed in [20], is shown in Fig. 3. The size of a typical memristor is relatively small, since the fabrication process is similar to processing the cross-layer via between metal layers. Memristors therefore exhibit high density and good scalability. The read and write time for these devices can be as fast as 120 picoseconds [25]. Currently, except for STT-MRAM, memristors suffer from endurance limitations, where the number of allowed writes per cell is approximately 10^{10} [26]. It is believed, however, that this limit will increase to at least 10^{15} [27]. Memristors may therefore solve many significant problems in the semiconductor industry, providing nonvolatile, dense, fast, and power-efficient memory.

III. IMPLY Logic Gate

The logic function p→q or p IMPLY q (also known as p IMPLIES q, material implication, and if p then q) is described in [7] and a truth table is listed in Table I. The IMPLY logic

TABLE I
TRUTH TABLE OF IMPLY FUNCTION

Case	*p*	*q*	*p→q*
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

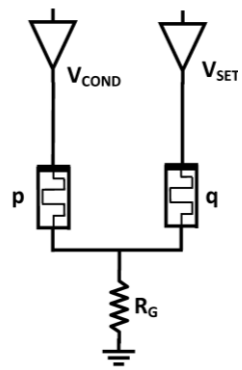


Fig. 4. IMPLY logic gate. The initial state of memristors p and q is the input of the logic gate and the output is the final state of the memristor q after applying the voltages V_{SET} and V_{COND}. A load resistor R_G is connected to both memristors.

function together with FALSE (a function that always yields the value zero as an output) comprises a computationally complete logic structure. Since the IMPLY function can be integrated within a memristor-based crossbar, IMPLY logic provides a basic logic element for a memristor-based circuit.

A. Basic Logic Gate Operation

The proposed memristor-based IMPLY logic gate uses a resistor R_G ($R_{\text{ON}} < R_G < R_{\text{OFF}}$) connected to two memristors, named P and Q, acting as digital switches. The corresponding initial memristances p and q are the inputs of the gate; while the output of the gate is the final memristance of Q (the result is written into the logic state q). Note that the memristance of both memristors changes during operation, i.e., the computation is destructive to both inputs. A schematic of an IMPLY gate is shown in Fig. 4.

The basic concept is to apply two different voltages to P and Q, where V_{SET}, the applied voltage on Q, has a higher magnitude than V_{COND}, the applied magnitude on P ($|V_{\text{COND}}| < |V_{\text{SET}}|$). If $p = 1$ (low resistance), the voltage on the common terminal is approximately V_{COND} and the voltage on the memristor Q is approximately $V_{\text{SET}} - V_{\text{COND}}$, which is sufficiently small to maintain the logic state of q. In the case of $p = 0$ and $q = 0$ (high resistances), the applied voltage on Q is approximately V_{SET} and Q is switched ON ($q = 1$). In the case of $p = 0$ and $q = 1$, the logic state of q is maintained. The memristance of an ideal IMPLY logic gate (zero delay time) for input cases 1 and 3 is shown in Fig. 5.

B. Analyzing the Behavior of a Logic Gate

V_{SET} and V_{COND}, the applied voltages on P and Q, are fixed. For any initial state, the memristor state q tends to drift

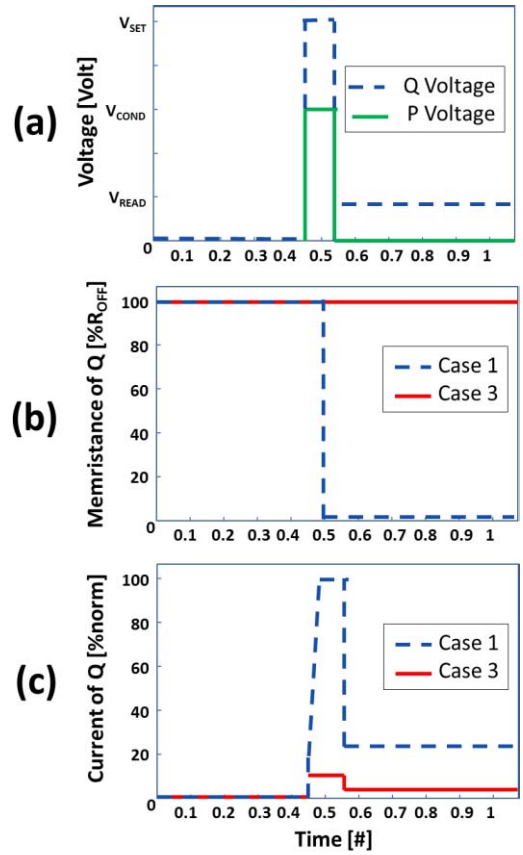


Fig. 5. Behavior of an ideal IMPLY logic gate. (a) Applied voltages on both memristors P and Q. (b) Memristance of Q for cases 1 and 3. While the memristance in case 1 decreases to R_{ON} within a zero write time, the memristance in case 3 does not change. (c) Current of memristor Q. The current in case 1 is sufficiently high to decrease the resistance of Q.

toward the ON state. For digital operation, the state of q should either stay unchanged or switch fully ON (changing the logic state from logical zero to logical one).

The different input combinations are listed in Table I. Due to the polarity of the memristors and the applied voltages, the memristance of memristor Q can only be reduced. Note that in cases 2 and 4, the initial logic state of q is logical one and the logic gate output q is also logical one. The gate operation, therefore, electrically reinforces the logic state of q since the memristance of Q is reduced.

In case 1, the initial state of q is logical zero; after applying the external voltages, q is switched ON. This case determines the time required to apply V_{SET} and V_{COND} until the logic state of q reaches the desired state (above a certain level of conduction that maintains correct logical behavior). This case determines the write time of the circuit (the delay time of the logic gate).

In case 3, the initial state of q is logical zero. This logic state should remain unchanged after applying V_{SET} and V_{COND}, although the voltages tend to change the internal state of q toward the ON state of logical one. This phenomenon is state drift. The logical zero state of q, which is the output of the gate, is electrically weaker than the input logical state of q (the memristance of Q after applying the voltages is lower than the initial memristance). State drift may require

refreshing the state; otherwise, repeated or prolonged sensing action may incorrectly switch the logic state of q. Note that the state drift phenomenon is a deterministic phenomenon. Stochastic switching [40], [41] can change the logical state of the memristors, and is not considered in this paper.

C. Speed–Robustness Tradeoff

The permissible value of the time required to apply V_{COND} and V_{SET} is determined from case 1. This write time is the delay time of the logic gate and determines the performance of the logic gate. Since the initial logical state of the memristors is unknown during operation (no preliminary read operation is applied), the voltages are applied at the same time for all input cases.

The state drift is determined from case 3, which depends upon the write time determined for case 1. Furthermore, any improvement in the performance due to changes in the applied voltage increases the state drift and degrades the robustness of the logic gate [28].

D. Extended Logic Functions Based on IMPLY

Any general Boolean function $f{:}B^n \rightarrow B$ can be implemented with only $n+3$ memristors [29], where three additional memristors carry out the computation. Only two memristors are required for up to three inputs. Computation of the function is performed in steps. In each step, either FALSE is applied to one memristor, or an IMPLY is applied to two memristors, where the output is written to a memristor (which is one of the inputs of the computational IMPLY stage). This process requires a long sequence of operations depending upon the number of inputs. This methodology has been improved in [30], where only two additional memristors are used rather than three. While a general algorithm to compute any Boolean function with a minimal number of memristors has been developed [29], [30], the computational process requires a large number of functional stages, and therefore requires significant computational time.

The schematic and sequence of a two input NAND, based on a memristor-based IMPLY gate and a FALSE logic gate, are shown in Fig. 6. This NAND gate is designed to minimize the computational time and number of memristors and is comprised of three memristors. The operation of this NAND logic gate changes the function of each memristor during the computing process. Two memristors act as inputs in the initial stage, one memristor acts as the output in the last stage, and all memristors act together as a computational logic element (as a memristor-based IMPLY gate) during different stages of the computing process. This application requires three computing stages (one FALSE and two IMPLY).

The IMPLY logic gate can also be extended to a multiple input NOR logic gate [31]. In this extension, as illustrated in Fig. 7(a), k input memristors $P_1, P_2, \ldots, P_k$, and a separate output memristor Q are assumed. The operation of this NOR gate requires two computational stages, the first stage initializes Q to logical zero ($q = 0$) and the second stage applies V_{SET} and V_{COND} in a manner similar to regular IMPLY. The extended NOR suffers from low fan-in since R_G needs to be scaled to all possible number of inputs. To solve this issue, a different structure has been proposed where a load resistor R_G is connected to every memristor and the load resistance varies, as shown in Fig. 7(b).

Step	Voltages	
Step 1: $s=0$		$V_S = V_{CLEAR}$
Step 2: $p \rightarrow s$	$V_P = V_{COND}$	$V_S = V_{SET}$
Step 3: $q \rightarrow s$	$V_q = V_{COND}$	$V_S = V_{SET}$

(a)

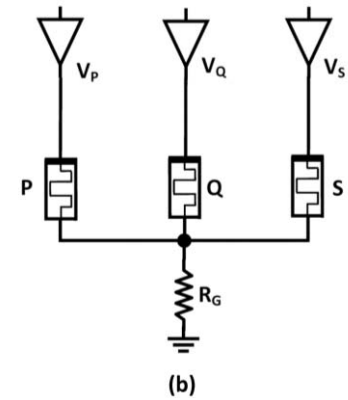
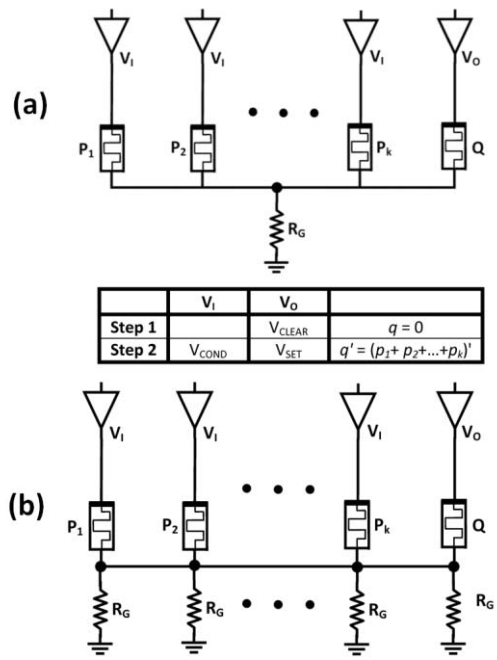


Fig. 6. IMPLY NAND, (a) The logic gate requires three sequential steps. (b) Schematic of IMPLY-based NAND gate.



	V_I	V_O	
Step 1		V_{CLEAR}	$q = 0$
Step 2	V_{COND}	V_{SET}	$q' = (p_1 + p_2 + \ldots + p_k)'$

Fig. 7. Extension to IMPLY, a k-input NOR. (a) Schematic based on execution of multiple implications in a single step and (b) improved fan-in structure, where the load resistors are dedicated to the participating logic devices.

IV. IMPLY Inside a Memristor-Based Crossbar

The IMPLY logic gate cannot be easily integrated with standard CMOS logic since both circuit structures are significantly different. In the IMPLY logic family, a resistance, rather than a voltage, represents the logical state. Furthermore, to operate the logic gate, a sequence of specific voltages is

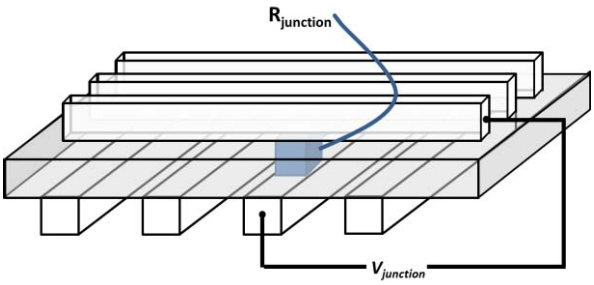


Fig. 8. Basic structure of a memristor-based crossbar. Each junction of the parallel lines is a memory cell with varying resistance R_{junction}.

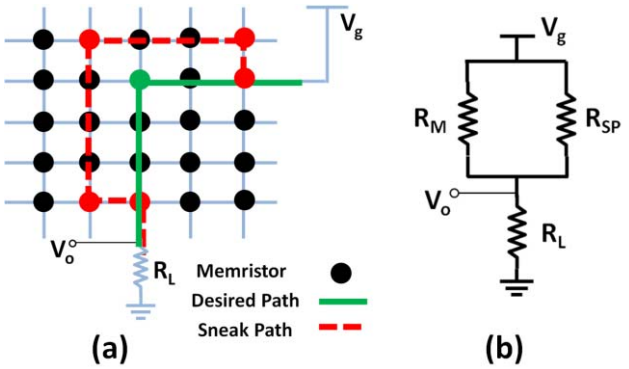


Fig. 9. Sneak path in a memristive crossbar. (a) Example sneak path. Every node in the grid is a memristor. The desired path is marked by a solid line and a sneak path is marked by a dashed line. (b) Equivalent circuit. All sneak paths have an equivalent resistance R_{SP} connected in parallel to the resistance of the memristor R_M.

applied to the memristors. The IMPLY logic gate therefore requires several computational stages (usually a different computational stage is executed during each clock cycle), and a separate mechanism to read the result of the computation and control the voltages. To integrate the IMPLY logic gate with standard voltage-based CMOS logic, a conversion mechanism is required. This mechanism includes a sense amplifier as well as additional components. The additional circuitry reduces the efficiency of integrating CMOS with a memristor-based IMPLY logic gate.

Alternatively, the IMPLY logic gate can be integrated inside a memristor-based crossbar array, commonly used for memory, where the input and output are values stored in the memory cells. This integration reduces power and provides an opportunity for novel non-von Neumann architectures. In this section, the basic structure of a memristor-based crossbar is presented, and a version of the IMPLY logic gate is illustrated.

A. Memristor-Based Crossbar

The basic structure of a memristor-based crossbar consists of two sets of parallel conductive (metal) lines. The conductive lines are perpendicular and behave as top and bottom electrodes to the memristive material, located between the lines [33]. The basic structure of a memristor-based crossbar is shown in Fig. 8. The write operation to a cell within the crossbar is achieved by applying a specific voltage to the junction, where a voltage is applied to both lines. For example, to write a logical one (low resistance), a positive voltage is applied to the column line and ground is connected to the row line (a positive voltage is applied to the memristor). To write a logical zero (high resistance), the column line is connected to ground and a positive voltage is connected to the row line (a negative voltage is applied to the memristor). These voltages are sometimes called V_{SET} (positive voltage to write a logical one, not necessarily the same voltage as in IMPLY) and V_{RESET} (negative voltage to write a logical zero). Since memristors are nonvolatile, the data does not change when no voltage is applied to the lines. The crossbar structure allows the density of the memory to be relatively high, since CMOS transistors are not used for each memory cell, but rather only to select the line. This memory structure is more than 20 times denser than DRAM [34].

The read operation of the crossbar is achieved by applying a relatively low voltage (e.g., lower than V_{SET}) to a junction and measuring the current. From Ohm's law, the resistance of the memristor is determined from this measured current. The current measurement is usually achieved by converting the current into a voltage through a voltage divider with a known resistance R_{pu}. The sensed voltage v_s is compared to a known voltage.

An undesired phenomenon in crossbars is sneak paths [35]–[38], which are undesired paths for the current flow. When a voltage is applied to a junction in the crossbar, current also flows through paths different than the desired path. These paths cross more than one memristor and add a resistance in parallel to the resistance of the memristor in the junction being read. An illustration of the sneak path phenomenon is shown in Fig. 9. This parallel resistance depends upon the stored data in the memristors in the undesired paths and changes the sensed voltage v_s from a simple voltage divider between R_{pu} and the resistance of the memristor to a voltage divider between R_{pu} and the total resistance of all memristors in all paths. A practical sensing operation should therefore consider all possible sneak paths. A schematic of a crossbar, including the read and write mechanisms, is depicted in Fig. 10. Several approaches exist to eliminate or reduce sneak paths, e.g., grounding inactive rows. In this paper, it is assumed that these approaches are used.

B. IMPLY in a Crossbar

The IMPLY logic gate can be integrated inside a crossbar, where P and Q are two memristors in the same row within the crossbar. The voltages V_{SET} and V_{COND} are the voltages of the word line, and the bit line is connected to a resistor R_G. To compute different Boolean functions with more than two memristors, the memristors are placed within the same row within the crossbar. Since the IMPLY operation is destructive to P and Q, if the data of the input to P is significant, a copy is assigned to a designated memristor. A schematic of a crossbar-based IMPLY logic gate is shown in Fig. 11.

V. Logic Gate Design Methodology

In this section, design considerations and constraints for a memristor-based IMPLY logic gate in a crossbar are described.

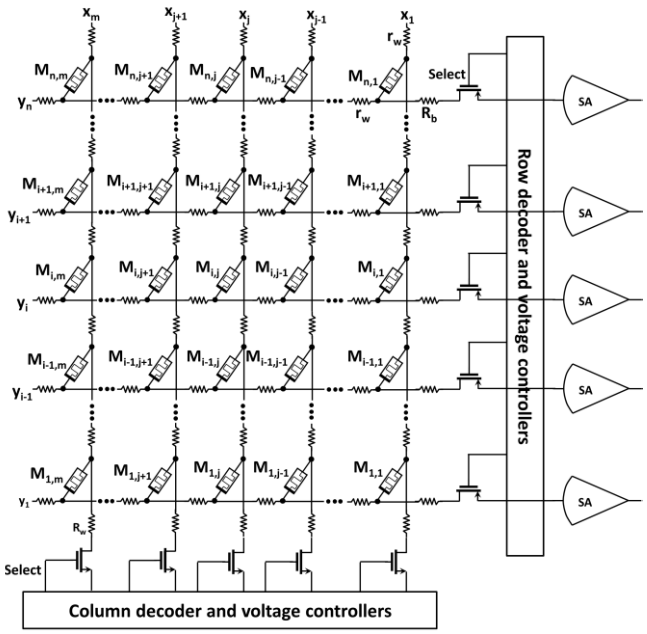


Fig. 10. $m \times n$ memristive crossbar. The columns show the word lines and the rows identify the bit lines. Each $M - ij$ is a memristor. The resistance of the conductive line is nr_w for the column line and mr_w for the row line. R_w and R_b are, respectively, the word and bit line resistance.

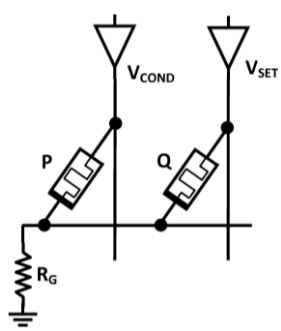


Fig. 11. IMPLY logic gate inside a memristor-based crossbar.

It is assumed that the memristor behavior is deterministic, rather than stochastic.

A. Design Flow and Constraints

Although no complete and accurate memristor model yet exists, all of the proposed memristor models are relatively complicated and the exact behavior of a memristive logic circuit is therefore mathematically cumbersome. A need therefore exists for heuristics for designing memristive circuits. For memristor-based IMPLY logic gates, the appropriate circuit parameters (R_G,V_{SET}, V_{COND}, and the time to apply the voltages T) need to be determined under some general constraints. These constraints include minimizing power consumption (only dynamic power consumption in a memristor-based crossbar), reducing area (the number of active memristors in a crossbar and the number of transistors in the controller), lowering the delay time of the logic gate, and increasing the robustness of the circuit (by reducing resistance drift during operation for those input cases where the logical output does not change). The parasitic capacitance of the CMOS transistors connected to the crossbar and the parasitic resistance of the metal lines as well as the sneak path phenomenon also need to be considered.

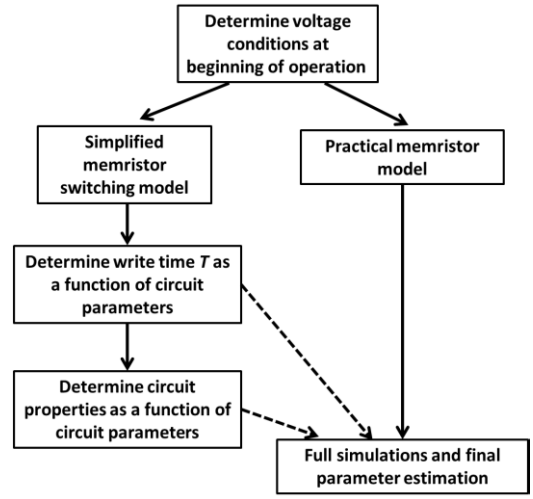


Fig. 12. Design flow for memristor-based IMPLY logic gates.

TABLE II
INPUT GATE VOLTAGES V_Q AND V_P, RESPECTIVELY, AT MEMRISTORS P AND Q AT $t = 0$, UNDER THE ASSUMPTIONS THAT THE MEMRISTANCE OF LOGIC ONE AND LOGIC ZERO IS, RESPECTIVELY, R_{ON} AND R_{OFF}, WHERE $R_{\mathrm{OFF}} \gg R_{\mathrm{ON}}$

Case	$V_Q(t=0)$	$V_P(t=0)$
1	$\frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_G}{R_{OFF}+2R_G}\cdot V_{COND}$	$-\left[\frac{R_G}{R_{OFF}+2R_G}\cdot V_{SET} - \frac{R_{OFF}+R_G}{R_{OFF}+2R_G}\cdot V_{COND}\right]$
2	$V_{SET}\cdot\frac{R_{ON}}{R_{OFF}}\cdot\frac{R_{OFF}+R_G}{R_{ON}+R_G} \approx V_{SET}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+R_G} - V_{COND}\right]$
3	$V_{SET} - V_{COND}\cdot\frac{R_G}{R_{ON}+R_G}$	V_{COND}
4	$V_{SET}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_G}{R_{ON}+2R_G}$	$-\left[V_{SET}\cdot\frac{R_G}{R_{ON}+2R_G} - V_{COND}\cdot\frac{R_{ON}+R_G}{R_{ON}+2R_G}\right]$

A general flow for the design of a memristor-based IMPLY logic gate is shown in Fig. 12. The design of a general Boolean function is demonstrated through a case study in Section VI. After determining the topology of the circuit, the conditions at the beginning of operation need to be determined. These static conditions do not depend on the memristor model and provide necessary conditions for correct circuit behavior. Simplified memristor models use several heuristics to approximate the circuit characteristics. The TEAM model [23] is used here to estimate the circuit parameters.

B. Design Constraints and Parameter Determination for IMPLY Logic Gate

In the design of a basic IMPLY logic gate, the circuit parameters V_{SET}, V_{COND}, and R_G and the time to apply the voltages T need to be determined. The memristor parameters (R_{ON}, R_{OFF}, k_{ON}, k_{OFF}, α_{ON}, α_{OFF}, i_{ON}, and i_{OFF} in the TEAM model) are fixed for a given technology.

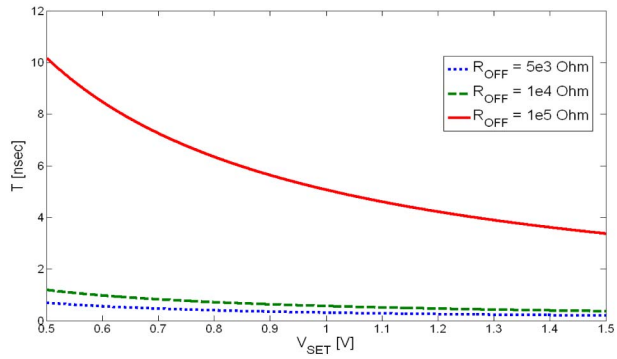


Fig. 13. Allowed write time T in case 1 for three values of R_{OFF} (5, 10, and 100 kΩ) under the assumptions of a binary resistance model and $Q' = 5 \times 10^{-14}$C.

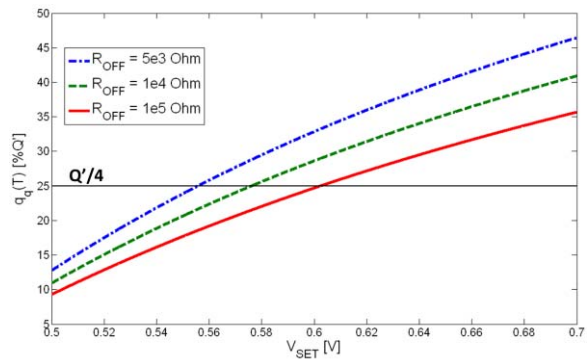


Fig. 14. Allowed values of V_{SET} for limited state drift in case 3 of $Q'/4$. V_{SET} is allowed if $q_q(T)$ is smaller than $Q'/4$ (horizontal line).

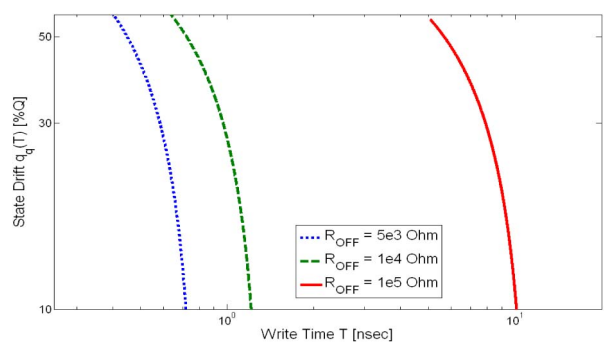


Fig. 15. Tradeoff between the speed (write time) and robustness (the state drift in case 3 for memristor Q) for three values of R_{OFF} (5, 10, and 100 kΩ) under the assumptions of a binary resistance model and $Q' = 5 \times 10^{-14}$C.

Although difficult to compute the time evolution of the voltage at Q (Fig. 4), it is possible to determine the voltage at Q at the beginning of the logic gate activity. The initial applied voltage at Q is different for each input case (a different initial memristance for Q and P). The initial voltages at P and Q are listed in Table II under the assumptions that the memristance of the logic one and logic zero is, respectively, R_{ON} and R_{OFF}, where $R_{\mathrm{OFF}} \gg R_{\mathrm{ON}}$.

From the initial applied voltages, some necessary conditions for correct logic behavior can be determined. The basic design principle is that the write (delay) time of the logic gate is determined from input case 1 (see Table II), but the circuit should also not exceed a specific state drift in input case 3.

A useful switching model is a binary memristance model [28]. Assume only two allowed memristances, R_{ON} and R_{OFF}. A total charge Q' flows through the memristor to cause the memristance R_{OFF} to switch to memristance R_{ON}. Under these assumptions and by solving both the switching behavior in case 1 and the write time T as a function of Q', the circuit parameter T is

$$T = \left[\frac{R_{\mathrm{OFF}}^2 + 2R_{\mathrm{OFF}}R_G}{R_{\mathrm{OFF}}V_{\mathrm{SET}} + R_G\left[V_{\mathrm{SET}} - V_{\mathrm{COND}}\right]}\right] \cdot Q'. \tag{5}$$

The write time for different circuit parameters and varying V_{SET} is shown in Fig. 13. Note that the logic gate is faster with a higher applied voltage or a smaller R_{OFF}.

Under this model, it is possible to limit the state drift (case 3 in Table II) for a fixed drift. The state drift is

$$q_q(T) \approx \left[V_{\mathrm{SET}} - \frac{R_G}{R_{\mathrm{ON}} + R_G}V_{\mathrm{COND}}\right] \cdot \left[\frac{R_{\mathrm{OFF}} + 2R_G}{R_{\mathrm{OFF}}V_{\mathrm{SET}} + R_G\left[V_{\mathrm{SET}} - V_{\mathrm{COND}}\right]}\right] \cdot Q' \tag{6}$$

where $q_q(T)$ is the total charge flowing through memristor Q after time T, as in case 3. If the state drift is limited to a value of $Q'/4$ as the maximum state drift, after four executions of the logic gate in case 3 the state drift would change the memristive logic state of q. This phenomenon requires a refresh every three executions of the logic gate since the logic state would change to an invert value during the fourth time. The allowed value of V_{SET} for several circuit parameters is shown in Fig. 14. Note that the state drift is more significant with a higher applied voltage, or with a smaller R_{OFF}. Combining Figs. 13 and 14, the tradeoff between the speed and robustness of a memristive IMPLY logic gate is illustrated in Fig. 15.

Another simple and useful memristor model assumes nonlinear behavior with a fixed threshold voltage V_{ON} [28]. Under this model, for an applied voltage below V_{ON}, the memristance is unchanged. To produce correct logical behavior, the initial applied voltage on Q must be above the threshold voltage in case 1 and below the threshold voltage in case 3. Adding this assumption to the initial applied voltage (see Table II) leads to the following two conditions on the circuit parameters:

$$R_{\mathrm{ON}} \cdot \frac{V_{\mathrm{SET}} - V_{\mathrm{ON}}}{V_{\mathrm{ON}} - \left[V_{\mathrm{SET}} - V_{\mathrm{COND}}\right]} < R_G < R_{\mathrm{OFF}} \cdot \frac{V_{\mathrm{SET}} - V_{\mathrm{ON}}}{2V_{\mathrm{ON}} - \left[V_{\mathrm{SET}} - V_{\mathrm{COND}}\right]} \tag{7}$$

$$\frac{V_{\mathrm{SET}}}{V_{\mathrm{COND}}} < \frac{R_{\mathrm{OFF}}}{R_{\mathrm{ON}}}. \tag{8}$$

The allowed value for R_G for several circuit parameters with varying V_{SET} is shown in Fig. 16. A reasonable value of R_G is the geometric mean of R_{ON} and R_{OFF}

$$R_G = \sqrt{R_{\mathrm{ON}} \cdot R_{\mathrm{OFF}}} \tag{9}$$

to maintain a constant ratio between each pair of resistances, R_{ON} and R_G, and R_G and R_{OFF}. Other values of R_G are also possible.

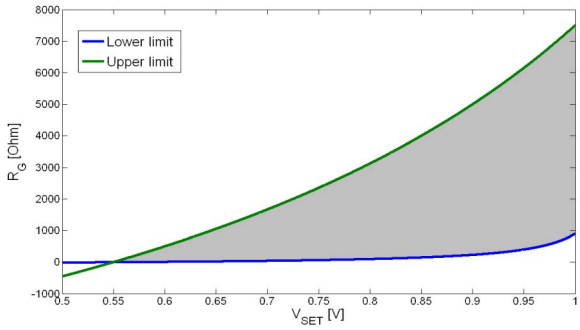


Fig. 16. Allowed value of R_G depends on V_{SET}. The upper line is the upper bound for allowed R_G and the lower line is the lower allowed bound for R_G. Under the assumption of a threshold voltage $V_{\text{ON}} = 0.55$ V, $V_{\text{COND}} = 0.5$ V, $R_{\text{ON}} = 100\ \Omega$, and $R_{\text{OFF}} = 10$ kΩ.

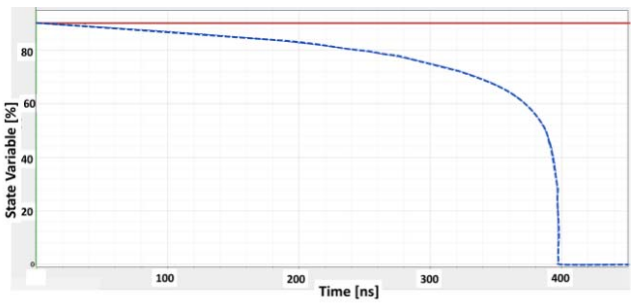


Fig. 17. State variable of q when applying an IMPLY logic gate for cases 1 (dashed line) and 3 (solid line). The parameters of the circuit are $V_{\text{SET}} = 1$ V, $V_{\text{COND}} = 0.5$ V, and $R_G = 10$ kΩ. The parameters of the memristors are $k_{\text{ON}} = 0.05$, $i_{\text{ON}} = 7\mu$A, and $\alpha_{\text{ON}} = 3$. The delay of the IMPLY logic gate is 397.1 ns and the state drift is 0.0007%, equivalent to 145,000 executions before the need to refresh.

C. Example of 1-Bit IMPLY Logic Gate

As a specific example of applying the flow chart of Fig. 12, assume the requirement is a maximum write time (delay) of 0.5 μsec. Note that the actual write time of a practical memristor is significantly faster [25]. The maximum allowed state drift is 0.00001 R_{OFF} (0.001% of the state drift as compared to full switching, equivalent to 10^5 executions of the logic gate before completely switching).

Assume a memristor with R_{ON} and R_{OFF}, respectively, of 1 and 100 kΩ. Set one circuit parameter V_{COND} to 0.5 V. From Figs. 13 and 14, note that as V_{SET} rises, the logic gate write time T decreases and the gate response is faster; however, the state drift phenomenon is more significant. From (8)

$$0.5\ \text{V} < V_{\text{SET}} < 50\ \text{V}. \tag{10}$$

This expression only produces a lower bound on V_{SET}, since the upper bound is significantly higher than practical on-chip supply voltages. For a current-controlled memristor (e.g., TEAM model), it is unrealistic to determine an exact equivalent voltage threshold (which depends on the transient memristance of the device). A sufficient approximation for an equivalent threshold voltage is

$$V_{\text{ON}} = i_{\text{ON}} \cdot R_{\text{OFF}} \tag{11}$$

where V_{ON} is the voltage threshold, and i_{ON} is the current threshold. For a memristor with a current threshold of 7 μA, the equivalent voltage threshold is 0.7 volts. From (7), R_G is

$$1.5\ \text{k}\Omega < R_G < 33.3\ \text{k}\Omega. \tag{12}$$

The widely used linear ion drift memristor model [12], [23] is incompatible with IMPLY logic gates. In this model, the memristance changes linearly for any applied voltage; the state drift phenomenon is therefore significant and intolerable for IMPLY logic gates [28]. Hence, a different memristor model with a current threshold, such as the TEAM model [23], is preferable. The TEAM model accurately describes the physical behavior of memristors. The chosen circuit parameters for this example are $R_{\text{ON}} = 1$ kΩ, $R_{\text{OFF}} = 100$ kΩ, $V_{\text{COND}} = 0.5$ V, $V_{\text{SET}} = 1$ V, and $R_G = 10$ kΩ. SPICE simulation based on these parameters for the memristance of q are shown in Fig. 17, where the write time (delay) of this logic gate is 397.1 ns and the state drift is 0.00069%, equivalent to about 145,000 executions before switching.

The write time (delay) and state drift for varying R_G and V_{SET} are listed in Tables III and IV. An increase in the resistance of R_G or decrease in the voltage level of V_{SET} increases the delay of the gate, but lowers the state drift phenomenon (and vice versa). The write time (delay) and state drift for different memristor parameters are listed in Table IV. An increase in the nonlinearity of the memristors (α_{ON}) increases the delay of the gate, but lowers the state drift phenomenon (and vice versa). An increase in k_{ON} decreases the delay of the gate without changing the state drift phenomenon.

TABLE III
WRITE TIME AND STATE DRIFT FOR DIFFERENT VALUES OF R_G. ALL VALUES SATISFY (10) AND (12). V_{COND} IS SET TO 0.5 V, $K_{\text{ON}} = 0.05$, $I_{\text{ON}} = 7\ \mu$A, AND $\alpha_{\text{ON}} = 3$

R_G [kΩ]	*T [μsec]*	*State Drift [%R_{OFF}]*	*Writes Before Refresh [#]*
1	0.1307	0.4655	215
3.5	0.1782	0.00244	4.09E4
5	0.2144	0.00184	5.43E4
10	0.3971	0.00069	1.45E5
15	0.7472	0.0009	1.15E6
17.5	1.038	0.00001	1.743E7
20	1.46	0	∞
30	3.063	0	∞

TABLE IV
WRITE TIME AND STATE DRIFT FOR DIFFERENT VALUES OF V_{SET} AND MEMRISTOR PARAMETERS. ALL VALUES SATISFY (19) AND (12). USING THE SAME DEFAULT VALUES AS TABLE III. $R_G = 10$ KΩ

Parameter	*T [μsec]*	*State Drift [%R_{OFF}]*	*Writes Before Refresh [#]*
Base	0.3971	0.00069	1.45E5
$V_{SET} = 1.2$ V	0.0945	0.31208	320
$k_{on} = 0.1$	0.1986	0.00069	1.45E5
$k_{on} = 0.01$	1.9866	0.0007	1.44E5
$\alpha_{on} = 1$	0.1587	0.3669	273
$\alpha_{on} = 4$	0.7927	0.0004	2.52E5

TABLE V
RESISTANCE OF A CMOS DRIVER FOR 0.12 μm CMOS PROCESS

W [µm]	W/L	CMOS Driver Resistance [Ω]	Voltage Drop with a Load of 100 kΩ
0.13	1	12.8k	11.33%
0.3	2.3	6.4k	6.00%
0.5	3.8	3.8k	3.67%
0.75	5.8	2.5k	2.42%
1	7.7	1.8k	1.83%
1.3	10	1.4k	1.33%
2.5	19.2	708	0.67%
5	38.5	349	0.33%
10	76.9	173	0.17%
20	153.8	86	0.08%

TABLE VI
STATE DRIFT OF THE IMPLY LOGIC GATE WITH CMOS BUFFERS AS COMPARED TO IDEAL VOLTAGE SOURCES FOR VARIOUS W/L RATIO

W [µm]	W/L	Difference in the State
0.13	1	-0.000502%
0.3	2.3	-0.000150%
0.5	3.8	0.000009%
0.75	5.8	0.000053%
1	7.7	0.000059%
1.3	10	0.000056%
2.5	19.2	0.000038%
5	38.5	0.000021%
10	76.9	0.000011%
20	153.8	0.000006%

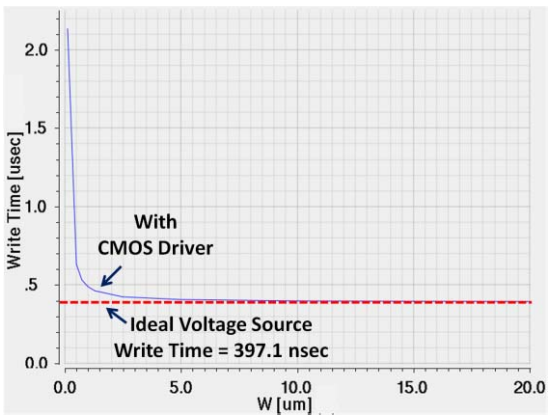


Fig. 18. Write time of an IMPLY logic gate with CMOS drivers for various CMOS widths (solid blue line) as compared to the write time with ideal voltage source (dashed red line). A 0.12 μm CMOS process is used; other circuit parameters are the same as in Fig. 17.

D. Variations in V_{SET} and V_{COND}

In previous sections, it is assumed that ideal voltage sources are used for V_{SET} and V_{COND}. Practical implementations, however, suffer from variations in the voltage level, mainly due to the resistance of the CMOS drivers. The CMOS drivers add resistance in series with the circuit and change the applied voltages. These voltage drops change the performance (as determined from input case 1) and the state drift (as determined from input case 3).

To evaluate the influence of CMOS drivers on performance and state drift, the IMPLY logic gate is simulated with similar circuit parameters as in Section V-C. The equivalent resistance of the CMOS driver for various CMOS widths is listed in Table V. The write time for different driver widths is shown in Fig. 18. For a W/L ratio of 10, the write time of the IMPLY logic gate with CMOS drivers increases by approximately 15%, as compared to ideal voltage sources. For a W/L ratio of 75, the increase in the write time is negligible (less than 1%).

To evaluate the change in the state drift phenomenon, the IMPLY logic gate is evaluated for input case 3. The difference in the state drift is listed in Table VI, exhibiting negligible difference for all W/L ratios. To overcome variations in the voltage source, the applied voltages (V_{SET} and V_{COND}) can be increased. Alternatively, the resistance of the circuit can be increased, by increasing R_G or using memristors with higher R_{ON} and R_{OFF} (e.g., the memristors in [42] have R_{ON} of approximately 300 kΩ), or the resistance of the CMOS driver can be decreased by increasing the W/L ratio.

VI. 8-BIT IMPLY FULL ADDER: A CASE STUDY

IMPLY together with FALSE (the function that always yields zero as an output) provide a complete logical structure. While any Boolean function can be executed, an efficient procedure is required to reduce the area and computational time. In this section, a case study of an 8-bit full adder is presented to discuss several design constraints and issues for general Boolean functions. In this case study, three approaches are considered: a general algorithm [29] is considered first, which requires a long sequence and only two additional memristors. Two other specific approaches–serial and parallel–are also considered. These approaches significantly reduce the required sequence of operational steps, where the parallel approach requires more memristors for faster execution as compared to the serial approach.

A. General Boolean Functions

An algorithm to implement any general Boolean function using only IMPLY and FALSE has been proposed in [29]. This algorithm requires $n + 3$ memristors for any general Boolean function $f: B^n \rightarrow B$. While this algorithm is efficient in terms of area (the number of memristors to compute a function), it is inefficient in terms of computational time and requires $O(2^{kn})$ computational steps, where n is the number of input memristors and k is the number of additional functional memristors for the computational process. A different approach is therefore required to improve the computational time. This new approach is demonstrated in this section through a case study.

Several Boolean functions being implemented by IMPLY and FALSE are listed in Table VII. These functions are the basic building blocks of any general Boolean function. Choosing the proper building blocks and computing sequence are key when the objective is to minimize the number of

TABLE VII
BASIC BOOLEAN OPERATIONS BASED ONLY ON IMPLY AND FALSE

Structure	*Operation*	*Comments*
0 → q	q' = 1	
1 → q	q' = q	
p → 0	q' = NOT(p)	
(A → (B → 0)) → 0	q' = A AND B	Result in different memristor than the inputs
(A → 0) → B	B' = A OR B	
(A → B) → ((B → A) → 0)	q' = A XOR B	Requires copying of the inputs, separate output *q*
FALSE(B), FALSE(C), A → C, C → B	B' = A	Copy operation – copy A to B

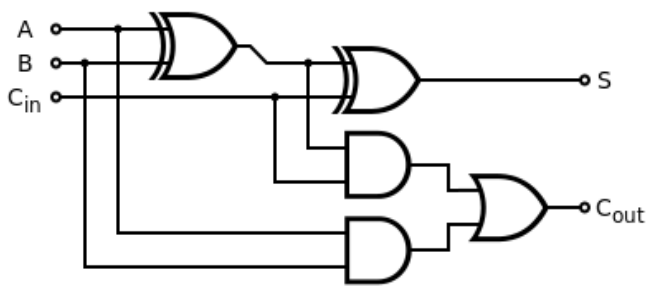


Fig. 19. Full adder consisting of two XOR gates, two AND gates, and an OR gate.

computational steps and memristors. To reduce the number of computational steps, parallelism can be exploited, where several IMPLY and FALSE operations occur during the same clock cycle. Since the operation is accomplished within the crossbar structure, the topology of the entire array needs to be considered, including possible sneak paths. Other methods for parallelism that do not suffer from sneak paths use unipolar memristors or, alternatively, insert switches between rows, which deviates from the crossbar structure. Modifying the crossbar structure to parallelize the execution is discussed in Section VI.

It is sometimes necessary to copy the value from a memory cell to other cells. The copy operation is also required when data is used multiple times, since the destruction of the input is undesired, or there is a need to transfer data to different rows within the crossbar. The copy operation is also listed in Table VII.

B. CMOS Full Adder

The input of the full adder are two 8-bit numbers and the output is one 8-bit number $S_7, S_6, \ldots, S_0$ and 1-bit carry C_{out}. The basic structure of a CMOS 8-bit ripple carry adder consists of eight full adders, where the logical operation of each adder is

$$S_i = A_i \oplus B_i \oplus C_i \tag{13}$$

$$C_{\text{out}} = (A_i \cdot B_i) + (C_i \cdot (A_i \oplus B_i)). \tag{14}$$

A single CMOS 8-bit adder consists of 400 CMOS transistors, as shown in Fig. 19 for a basic full adder.

TABLE VIII
COMPARISON OF N-BIT FULL ADDERS. THE NUMBERS IN THE BRACKETS ARE FOR AN 8-BIT FULL ADDER

		Base [29]	Optimized Approaches	
			Serial	Parallel
Execution steps		89*N* (712)	29*N* (232)	5*N*+18 (58)
Memristors	Input	2*N*	2*N*	2*N*
	Output	*N*+1	*N*+1	*N*+1
	Functional	4	2	6*N*-1
	Total	3*N*+5 (29)	3*N*+3 (27)	9*N* (72)
Special functions required	Parallel FALSE	-	-	V
	IMPLY between lines	-	-	V
	TRUE	V	-	-

C. IMPLY Full Adder

Several approaches exist to design an 8-bit full adder based solely on IMPLY and FALSE operations. The basic approach is to follow the algorithm proposed in [29]. Two additional approaches are considered–serial and parallel. To evaluate these approaches, the total number of memristors and the number of computation steps are compared. The general algorithm from [29] requires 712 computational steps, while the serial approach lowers the computational time to 232 computational steps with approximately the same number of memristors, and the parallel approach has the best performance of 58 computational steps but requires double the number of memristors. A comparison among the approaches is listed in Table VIII.

To execute a XOR operation, two functional memristors *M1* and *M2* are required, where the complete sequence, as listed in Table VII, is

$$\begin{aligned} A \text{ XOR } B: \quad & \text{FALSE}(M1), \text{FALSE}(S), A \to S, S \to M1 \\ & \text{FALSE}(M2), \text{FALSE}(S), B \to S, S \to M2 \\ & B \to M1, \text{FALSE}(S), M1 \to S \\ & A \to M2, M2 \to S. \end{aligned}$$

The first two rows are copy operations of A and B, respectively, to *M1* and *M2* since the IMPLY operation destroys both inputs. To execute S_i, the execution process is divided into two XOR operations, where (13) is

$$S_i = (A_i \oplus B_i) \oplus C_i. \tag{15}$$

This execution requires two functional memristors and 26 computational steps for S_i, while the intermediate XOR of A_i and B_i is also used for $C_{\text{out},i}$, where (14) becomes

$$\begin{aligned} C_{\text{out},i} = & \left(A_i \to \left(B_i \to' 0'\right)\right) \\ & \to \left(\left(C_i \to \left(\left(A_i \oplus B_i\right) \to' 0'\right)\right) \to' 0'\right). \end{aligned} \tag{16}$$

Several possible sequences exist for executing C_i using three functional memristors to decrease the number of computational steps. Furthermore, A_i, B_i, and C_i can also be treated as functional memristors after the initial value is changed during the execution process. The complete sequence is described in the supplementary material.

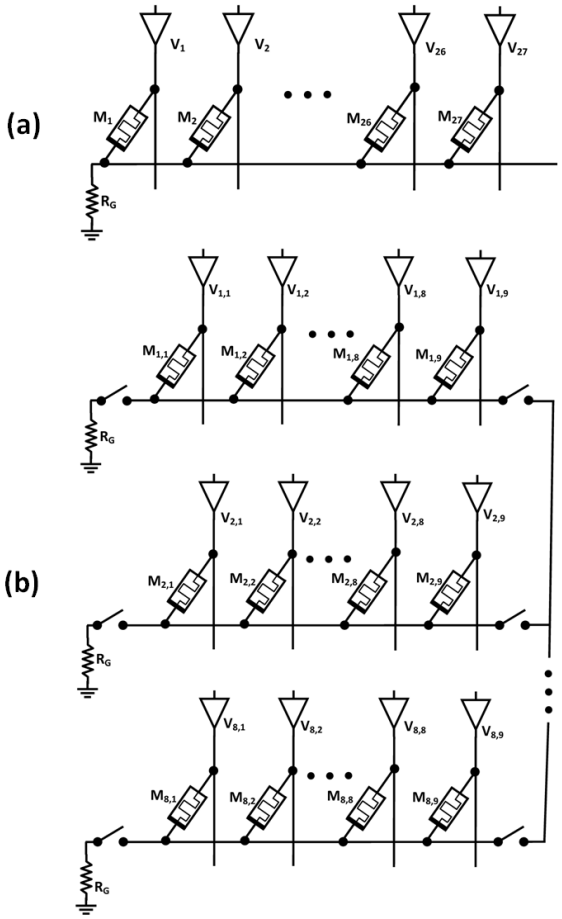


Fig. 20. 8-bit full adder for (a) serial approach and (b) parallel approach. For the serial approach, 27 memristors are used in the same row of a standard crossbar structure. The parallel approach requires a more complex crossbar structure, where a switched connection between rows exists. Each bit execution is done in a different row using nine memristors.

For an 8-bit full adder, two approaches have been examined in the case study. The serial approach executes one operation every clock cycle–IMPLY or FALSE. For the serial approach, all memristors are in the same row, as shown in Fig. 20(a). In the parallel approach, independent operations are executed during the same clock cycle, reducing the number of required computational stages. For the parallel approach, each bit in the full adder is in a different row, as shown in Fig. 20(b). The carry is passed between the different rows and the FALSE operations are simultaneously completed for several memristors. The parallel approach requires some modifications which differ from the crossbar structure, adding connections between the rows of the crossbar. These modifications also eliminate the sneak path phenomenon while increasing the area as compared to a conventional crossbar.

VII. Beyond von Neumann: Logic Inside the Memory

IMPLY logic is a natural method to execute logical operations within the memristors. Memristor-based IMPLY logic has the same crossbar structure as a memristor-based memory and therefore enables the capability of performing logic operations inside the memory with the same cells used to store data. This combination enables innovative computing architectures, rather than the classical von Neumann architecture where the computing operations and the data storage are separated.

For these novel architectures, part of the computation is achieved inside the memory, with no separation with the data read and write operations. These architectures are particularly appropriate for massive parallel applications, where vast amount of data need to be processed. In von Neumann architecture for massive parallel applications, the data transfer requires a wide data bus, long latency, and consumes relatively high power. In these novel architectures, the memory and logical operations are in the same crossbar structure, almost no data transfer is required, and the latency and power are significantly reduced, although the memristor IMPLY logic delay is greater than the CMOS logic delay.

In these innovative architectures, the memristive memory serves two roles–as memory to store data and as a computational unit. The function of a specific memristor can be decided dynamically. Each memristor can act as either a memory cell or as part of an IMPLY logic gate in different stages of the operation. The effective size of the memory and the computational unit is flexible and can vary for different applications. A memristor-based memory requires a relatively complex controller that behaves as a regular memory controller and also sends control signals (V_{SET} and V_{COND}) to the IMPLY logic gates. This novel architecture requires a new instruction set, requiring specific instructions for logic operations inside the memory.

VIII. Conclusion

An IMPLY logic gate is a natural way to perform logic operations with memristors. This logic gate can be integrated within a memristor-based memory and, together with FALSE, provide a complete logic family. This memristive logic gate also enables non-von Neumann architectures, which may open a new era in computer architecture.

The potential benefits of memristive circuits in terms of density and power support further work in this field. The results described in this paper can be used to direct further research on device structure optimization, logic synthesis methods, array structures, and computing architectures.

References

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

[2] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, Feb. 1976.

[3] 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, Jun. 2005.

[4] S. Kvatinsky, N. Wald, G. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Hybrid CMOS-memristor logic," *IEEE Trans. Very Large Scale Integr. (VLSI)*, in preparation.

[5] M. Klimo and O. Such, *Memristors Can Implement Fuzzy Logic*. Ithaca, NY, USA: Cornell Univ. Press, Oct. 2011.

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

[7] 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, pp. 873–876, Apr. 2010.

[8] Y. V. Pershin and M. Di Ventra, "Neuromorphic, digital and quantum computation with memory circuit elements," *Proc. IEEE*, vol. 100, no. 6, pp. 2071–2080, Jun. 2012.

[9] 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.

[10] D. Biolek, Z. Biolek, and V. Biolkova, "Pinched hysteresis loops of ideal memristors, memcapacitors, and meminductors must be 'self-crossing'," *Electron. Lett.*, vol. 47, no. 25, pp. 1385–1387, Dec. 2011.

[11] L. O. Chua, "Resistance switching memories are memristors," *Appl. Phys. A, Mater. Sci. Process.*, vol. 102, no. 4, pp. 765–783, Mar. 2011.

[12] 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.

[13] D. Sacchetto, M. H. Ben-Jamaa, S. Carrara, G. DeMicheli, and Y. Leblebici, "Memristive devices fabricated with silicon nanowire Schottky barrier transistors," in *Proc. IEEE Int. Symp. Circuits Syst.*, May/Jun. 2010, pp. 9–12.

[14] K. A. Campbell, A. Oblea, and A. Timilsina, "Compact method for modeling and simulation of memristor devices: Ion Conductor Chalcogenide-based Memristor Devices," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architect.*, Jun. 2010, pp. 1–4.

[15] X. Wang, Y. Chen, H. Xi, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE Electron Device Lett.*, vol. 30, no. 3, pp. 294–297, Mar. 2009.

[16] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, Jun. 2009.

[17] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A versatile memristor model with non-linear dopant kinetics," *IEEE Trans. Electron Devices*, vol. 58, no. 9, pp. 3099–3105, Sep. 2011.

[18] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, pp. 429–433, Jul. 2008.

[19] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *Proc. 12th Int. Workshop Cellular Nanosc. Netw. Appl.*, Feb. 2010, pp. 1–4.

[20] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *J. Appl. Phys.*, vol. 106, no. 7, pp. 1–6, Oct. 2009.

[21] H. Abdalla and M. D. Pickett, "SPICE modeling of memristors," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 1832–1835.

[22] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE Electron Device Lett.*, vol. 32, no. 10, pp. 1436–1438, Oct. 2011.

[23] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM—ThrEshold adaptive memristor model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 1, pp. 211–221, Jan. 2013.

[24] J. Borghetti, Z. Li, J. Strasnicky, X. Li, D. A. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proc. Nat. Acad. Sci. United States Amer.*, vol. 106, no. 6, pp. 1699–1703, Feb. 2009.

[25] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, no. 48, pp. 1–7, Nov. 2011.

[26] J. J. Yang, M.-X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams, "High switching endurance in TaO_x memristive devices," *Appl. Phys. Lett.*, vol. 97, no. 23, pp. 1–3, Dec. 2010.

[27] J. Nickel, "Memristor materials engineering: From flash replacement towards a universal memory," in *Proc. IEEE IEDM Adv. Memory Technol. Workshop*, Dec. 2011, pp. 1–3.

[28] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based IMPLY logic design procedure," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2011, pp. 142–147.

[29] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2009, pp. 33–36.

[30] E. Lehtonen, J. H. Poikonen, and M. Laiho, "Two memristors suffice to compute all boolean functions," *Electron. Lett.*, vol. 46, no. 3, pp. 239–240, Feb. 2010.

[31] 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.

[32] 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, Apr. 2010.

[33] A. Flocke and T. G. Noll, "Fundamental analysis of resistive nano-crossbars for the use in hybrid Nano/CMOS-memory," in *Proc. Eur. Solid State Circuits Conf.*, Sep. 2007, pp. 328–331.

[34] M. A. Zidan and K. N. Salama, "Memristor based memory: The sneak paths problem and solutions," *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, Feb. 2013.

[35] C. A. David and B. Feldman, "High-speed fixed memories using large-scale integrated resistor matrices," *IEEE Trans. Comput.*, vol. 17, no. 8, pp. 721–728, Aug. 1968.

[36] W. T. Lynch, "Worst-case analysis of a resistor memory matrix," *IEEE Trans. Comput.*, vol. 18, no. 10, pp. 940–942, Oct. 1969.

[37] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proc. IEEE*, vol. 100, no. 6, pp. 2021–2032, Jun. 2012.

[38] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 1–5.

[39] O. Kavehei, S. Al-Sarawi, K.-R. Cho, K. Eshraghian, and D. Abbot, "An analytical approach for memristive nanoarchitectures," *IEEE Trans. Nanotechnol.*, vol. 11, no. 2, pp. 374–385, Mar. 2012.

[40] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, P. Crozat, N. Zerounian, J.-V. Kim, C. Chappert, and H. Ohno, "Single-shot time-resolved measurement of nanosecond-scale spin-transfer induced switching: Stochastic versus deterministic aspects," *Phys. Rev. Lett.*, vol. 100, no. 5, pp. 057206-1–057206-4, Feb. 2008.

[41] R. Soni, P. Meuffels, G. Staikov, R. Weng, C. Kügeler, A. Petraru, M. Hambe, R. Waser, and H. Kohlstedt, "On the stochastic nature of resistive switching in Cu doped $Ge_{0.3}Se_{0.7}$ based memory devices," *J. Appl. Phys.*, vol. 110, no. 5, pp. 054509-1–054509-10, Sep. 2011.

[42] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba, and W. Lu, "Synaptic behaviors and modeling of metal oxide memristive device," *Appl. Phys. A*, vol. 102, no. 4, pp. 857–863, Feb. 2011.

Shahar Kvatinsky (S'12) received the B.Sc. degree in computer engineering and applied physics and the M.B.A. degree from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2009 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, Technion-Israel Institute of Technology, Haifa, Israel.

Prior to his Ph.D. studies, he worked for Intel as a circuit designer.

Guy Satat received the B.Sc. degree in electrical engineering and the B.Sc. degree in physics from the Technion-Israel Institute of Technology, Haifa, Israel, as part of the Technion's Program for excellent students.

He joined Intel, Inc., in 2011, and worked on interconnect architecture. In 2013, he joined the Media Laboratory, Camera Culture Group, Massachusetts Institute of Technology, Cambridge, MA, USA, as a Graduate Student, and worked on ultrafast imaging and health imaging.

Nimrod Wald received the B.Sc. degree in electrical engineering and physics from Technion-Israel Institute of Technology, Haifa, Israel, in 2013.

He joined Qualcomm, Inc., San Diego, CA, USA, in 2011, as a Hardware Designer, and he has been a Hardware Architect since 2013 in the area of performance analysis.

Eby G. Friedman (M'79–SM'90–F'00) received the B.S. degree from Lafayette College, Easton, PA, USA, in 1979, and the M.S. and Ph.D. degrees from the University of California, Irvine, CA, USA, in 1981 and 1989, respectively, all in electrical engineering.

He was with Hughes Aircraft Company, Glendale, CA, USA, from 1979 to 1991, rising to the position of manager of the Signal Processing Design and Test Department, responsible for the design and test of high performance digital and analog IC's. He has been with the Department of Electrical and Computer Engineering at the University of Rochester, Rochester, NY, USA, since 1991, where he is a Distinguished Professor, and the Director of the High Performance VLSI/IC Design and Analysis Laboratory. He is also a Visiting Professor with the Technion-Israel Institute of Technology. His current research interests include high performance synchronous digital and mixed-signal microelectronic design and analysis with application to high speed portable processors and low power wireless communications.

Dr. Friedman is the author of over 400 papers and book chapters, 12 patents, and the author or editor of 16 books in the fields of high speed and low power CMOS design techniques, 3-D design methodologies, high speed interconnect, and the theory and application of synchronous clock and power delivery. He is the Regional Editor of the *Journal of Circuits, Systems and Computers*, a member of the editorial boards of the *Analog Integrated Circuits and Signal Processing*, *Microelectronics Journal*, *Journal of Low Power Electronics*, *Journal of Low Power Electronics and Applications*, and IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, Chair of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS steering committee, and a member of the technical program committee of a number of conferences. He previously was the Editor-in-Chief of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, a member of the editorial board of the Proceedings of the IEEE, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, and Journal of Signal Processing Systems, a Member of the Circuits and Systems (CAS) Society Board of Governors, Program and Technical chair of several IEEE conferences, and a recipient of the University of IEEE CAS Charles A. Dosoer Technical Achievement Award, Rochester Graduate Teaching Award, and a College of Engineering Teaching Excellence Award. He is a Senior Fulbright Fellow.

Avinoam Kolodny (SM'11) received the Doctoral degree in microelectronics from the Technion-Israel Institute of Technology, Haifa, Israel, in 1980.

He joined Intel Corporation, where he was engaged in Research and Development in the areas of device physics, VLSI circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Eletrical Engineering, Technion since 2000. His current research interests include interconnects in VLSI systems at both physical and architectural levels.

Uri C. Weiser (F'02) received the bachelor's and master's degrees in electrical engineering from Technion, Haifa, Israel and the Ph.D. degree in computer science from the University of Utah, Salt Lake City, UT, USA.

He is a Visiting Professor with the Electrical Engineering Department, Technion IIT, and an Advisor at numerous startups. He was with Intel from 1988 to 2006. At Intel, he initiated the definition of the first Pentium processor, drove the definition of Intel's MMX technology, invented (with A. Peleg) the Trace Cache, he co-managed and established the Intel Microprocessor Design Center, Austin, TX, USA, and later initiated an advanced media applications research activity. He was appointed Intel Fellow in 1996. He was with the Israeli Department of Defense as a Research and System Engineer and with National Semiconductor Design Center, Israel, where he led the design of the NS32532 microprocessor.

Dr. Weiser was an Associate Editor of the *IEEE Micro Magazine* from 1992 to 2004 and *Computer Architecture Letters*. He was a fellow of ACM.