In-Memory Computing Using Paths-Based Logic and Heterogeneous Components

Alvaro Velasquez and Sumit Kumar Jha
Department of Computer Science, University of Central Florida, {velasquez, jha}@cs.ucf.edu

***Abstract*—The memory-processor bottleneck and scaling difficulties of the CMOS transistor have given rise to a plethora of research initiatives to overcome these challenges. Popular among these is in-memory crossbar computing. In this paper, we propose a framework for synthesizing logic-in-memory circuits based on the behavior of paths of electric current throughout the memory. Limitations of using only bidirectional components with this approach are also established. We demonstrate the effectiveness of our approach by generating n-bit addition circuits that can compute using a constant number of read and write cycles.**

I. Introduction

The subject of logic-in-memory crossbar, or crosspoint, computing has been reinvigorated in recent years. This is due, in part, to significant advances in memristor technology and the potential for low-power computing. As a result, several logic synthesis procedures have been proposed using crossbars [1] [2] [3] [4] [5] [6] [7]. In this paper, we present an approach to generate crossbar computing designs that compute a given Boolean formula. As an example, we demonstrate how n-bit addition can be computed in memory using $\mathcal{O}(1)$ number of read/write operations as opposed to the $\mathcal{O}(n)$ steps required by competing approaches (See Table I).

II. Related Work

In-memory crossbar computing often uses memristors, which can be thought of as charge-dependent programmable resistors, as the underlying circuit element. In this context, digital computing is largely based on the implication-falsity logic presented in [8]. We briefly present an outline of this procedure. Let P and Q denote two memristors with corresponding states $p, q \in \{0, 1\}$, where 0 and 1 denote high-resistance and low-resistance states (HRS, LRS), respectively. Both devices are connected to the same load resistor R_G and let $V_{\text{COND}}, V_{\text{TH}}, V_{\text{SET}}$ denote voltages such that V_{SET} is sufficient to switch a memristor to the LRS state, V_{TH} is the threshold voltage value that must be exceeded in order to switch said memristor, and $V_{\text{SET}} - V_{\text{COND}} < V_{\text{TH}}$. Apply voltage bias V_{COND} to P and V_{SET} to Q. This will cause Q to switch to or remain in the LRS state if $p \implies q$. See Fig. 1 for a visualization of the case when $p = 1$ and $q = 0$. It is easy to see that Q will be switched to or remain in the LRS state under the other 3 evaluations of p and q because the voltage drop across Q will be $V_{\text{SET}} > V_{\text{TH}}$. A voltage V_{CLEAR} can be used to set the memristor to the HRS state corresponding to a binary value of 0, or *false*. This combination of implication and falsity is functionally complete for Boolean operations. This result catalyzed a host of design methodologies [2], [3], [4], [5] based on minimizing the number of implication operations required to compute a formula of interest.

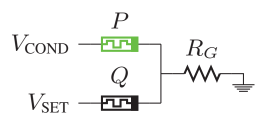


Fig. 1: Memristor implication logic. The green memristor P is in the LRS state ($p = 1$) and the black memristor is in the HRS state ($q = 0$). The voltage drop across Q will be $V_{\text{SET}} - V_{\text{COND}} < V_{\text{TH}}$, so Q will remain in the HRS state, meaning that $q = 0 = (p \implies q)$.

In [6] and [7], each interconnect is a complementary resistive switch consisting of two memristors of opposite polarity. The basic logical operation on these interconnects follows the equation $Z^t = (w \impliedby b) \wedge Z^{t-1} \vee (w \not\Rightarrow b) \wedge \neg Z^{t-1}$, where Z^t is the state of the device at time t, w (b) is the wordline (bitline) such that $w = 1$ ($b = 1$) in the presence of a high voltage potential and 0 otherwise. This allows greater control over the flows of current throughout the crossbar and it is demonstrated how under this approach an n-bit NAND operation can be computed in a constant number of steps.

III. Paths-Based Logic

Our approach utilizes the paths of current throughout the crossbar (See Definition 1) in order to compute a Boolean formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$, where $\mathbb{B}$ is the set $\{0, 1\}$. By methodically programming the crossbar components to be variables in ϕ, we can manipulate the trajectories of paths induced so that there is a flow of current between two specified wires if and only if ϕ holds. For the remainder of this paper, we utilize capital letters to signify wires and components and lowercase letters to denote their value. For example, we say that a wire W has value $w = 0$ if there is negligible current on the wire and $w = 1$ if there is a significant amount. Similarly, given $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$, we represent the set-theoretic version of the k^{th} formula ϕ^k as Φ^k, where Φ^k_i denotes the i^{th} clause of Φ^k and Φ^k_{ij} denotes the j^{th} variable in the i^{th} clause. For example, given $\phi : \mathbb{B}^2 \mapsto \mathbb{B}^2$, where $\phi^1 = (a \wedge b) \vee (\neg a \wedge \neg b)$ and $\phi^2 = (\neg a \wedge b) \vee (a \wedge \neg b)$, we represent these formulas as sets $\Phi^1 = \{\{A, B\}, \{\neg A, \neg B\}\}$ and $\Phi^2 = \{\{\neg A, B\}, \{A, \neg B\}\}$, respectively. This representation facilitates later proofs.

Definition 1 (Homogeneous Crossbar)**.** *A homogeneous crossbar is a tuple $\mathcal{X} = (M, R, C)$, where $M = (M_{ij})$ is the set of components such that $m_{ij} = 1$ ($m_{ij} = 0$) denotes an LRS (HRS) node. The sets $R = (R_i)$ and $C = (C_j)$ denote the sets of wordlines, or row wires, and bitlines, or column wires. A value of $r_i = 1$ ($r_i = 0$) denotes the presence (absence) of electric current in wire R_i. Let $W = R \cup C$ denote all wires.*

Axiom 1 (Homogeneous Flow). *Given a homogeneous crossbar $\mathcal{X} = (M, R, C)$, $(r_i \wedge m_{ij}) \implies c_j$ and $(c_j \wedge m_{ij}) \implies r_i$ always hold. As a result, property (1) holds.*

$$\bigwedge_{j=1}^{|C|} \left(c_j \iff \bigvee_{i=1}^{|R|} m_{ij} \wedge r_i \right) \wedge \bigwedge_{i=1}^{|R|} \left(r_i \iff \bigvee_{j=1}^{|C|} m_{ij} \wedge c_j \right) \quad (1)$$

As its name suggests, paths-based logic seeks to use the paths of current throughout the crossbar as a means of computation. An initial flow of current is generated by applying a voltage bias to some source wire and grounding another wire. A path is a sequence of nodes connecting two wires. For example, the path $\Pi^{R_i \to C_j} = (M_{ij_1}, M_{i_1 j_1}, M_{i_1 j_2}, M_{i_2 j_2}, \ldots, M_{i_k j})$ connects wires R_i and C_j, where $\pi_d^{R_i \to C_j}$ denotes the value of the d^{th} component. From Axiom 1, the following chain of implications holds.

$$\begin{aligned} (r_i \wedge m_{ij_1} \implies c_{j_1}) \wedge (c_{j_1} \wedge m_{i_1 j_1} \implies r_{i_1}) \wedge \\ (r_{i_1} \wedge m_{i_1 j_2} \implies c_{j_2}) \wedge \cdots \wedge (r_{i_k} \wedge m_{i_k j} \implies c_j) \end{aligned} \quad (2)$$

We can also show that $(c_j \wedge m_{i_k j} \implies r_{i_k}) \wedge \cdots \wedge (c_{j_1} \wedge m_{ij_1} \implies r_i)$. Thus, if every component in a path $\Pi^{W_i \to W_j}$ is in the LRS state, then current flowing in the source wire W_i will be redirected to the destination wire W_j, and vice-versa. We call this the symmetry property of paths. A general form of this property is captured by equation (3).

$$\left(w_i \wedge \bigwedge_d \pi_d^{W_i \to W_j} \implies w_j \right) \wedge \left(w_j \wedge \bigwedge_d \pi_d^{W_i \to W_j} \implies w_i \right) \quad (3)$$

We have seen that paths can be treated as a conjunction of variables mapped to components. Thus, it is convenient to think of formulas in their disjunctive normal form (DNF). Any Boolean formula can be written in DNF as a disjunction of conjunctive clauses (i.e. $\bigvee_i \bigwedge_j \phi_{ij}$). For the remainder of this paper, we assume that all Boolean formulas are in DNF.

Given a formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and a crossbar $\mathcal{X} = (M, R, C)$, we want to find a mapping of the components M to variables $b_1, \ldots, b_p$ in ϕ and constants $\{0, 1\}$ denoting the HRS and LRS states. We define this mapping by $P = (P_{ij})$, $P_{ij} \in \{B_1, \neg B_1, \ldots, B_p, \neg B_p\} \cup \{0, 1\}$. P defines the configuration of M as described by (4). See Fig. 2 for an example.

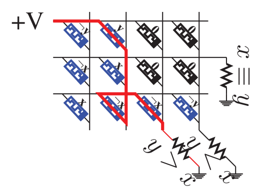
$$\begin{aligned} (m_{ij} = 1) \iff (P_{ij} = 1) \vee ((P_{ij} = B_k) \wedge b_k) \vee \\ ((P_{ij} = \neg B_k) \wedge \neg b_k) \end{aligned} \quad (4)$$

Definition 2 (Well-Formed Design). *Suppose we are given a Boolean formula $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and crossbar $\mathcal{X} = (M, R, C)$ with input and output wires $S, F \subset W$ such that the value of the k^{th} formula ϕ^k will be output to wire F_k. A well-formed design is a mapping matrix P such that there are paths $\Pi^{S_i \to F_j}$ that satisfy (5) for all evaluations $\alpha \in \mathbb{B}^p$ of ϕ.*

$$\bigwedge_{k=1}^{q} \left(f_k \iff \phi^k \right) \quad (5)$$

In order for (5) to hold, there must be paths from the source wires S to the destination wires F such that current will flow from source S_i to destination F_j if and only if ϕ^j is true. Let us elucidate this with an example. Suppose we want to find a well-formed design for a 1-bit comparator using a 3×4 homogeneous crossbar $\mathcal{X} = (M, (R_1, R_2, R_3), (C_1, C_2, C_3, C_4))$. Given two bits x and y, we want outputs indicating the three possible outcomes $(x \equiv y) = (x \wedge y) \vee (\neg x \wedge \neg y)$, $(y > x) = \neg x \wedge y$, or $(y < x) = x \wedge \neg y$. Let $S = (R_1)$ and $F = (R_2, C_3, C_4)$ denote the sets of source and destination wires, respectively. Suppose $s_1 = 1$ for all evaluations $\alpha \in \mathbb{B}^2$. This means that there will be a flow of current on S_1 regardless of the values in the evaluation vector. We want a well-formed design P that assigns values to each component M_{ij} so that there will be paths $\Pi^{R_1 \to R_2}$, $\Pi^{R_1 \to C_3}$, $\Pi^{R_1 \to C_4}$ from the source wire to the destination wires resulting in $r_2 \iff (x \equiv y)$, $c_3 \iff (y > x)$, and $c_4 \iff (y < x)$. For each $\alpha \in \mathbb{B}^2$, the paths are as follows (See Fig. 2).

- $\alpha = (x = 0, y = 0)$: $\Pi^{R_1 \to R_2} = (M_{11}, M_{21})$
- $\alpha = (x = 0, y = 1)$: $\Pi^{R_1 \to C_3} = (M_{12}, M_{32}, M_{33})$
- $\alpha = (x = 1, y = 0)$: $\Pi^{R_1 \to C_4} = (M_{11}, M_{31}, M_{34})$
- $\alpha = (x = 1, y = 1)$: $\Pi^{R_1 \to R_2} = (M_{12}, M_{22})$



$$P = \begin{array}{c} R_1 \\ R_2 \\ R_3 \end{array} \begin{pmatrix} \neg y & y & 0 & 0 \\ \neg x & x & 0 & 0 \\ x & \neg x & \neg x & \neg y \end{pmatrix} \quad (6)$$
$$\qquad\qquad C_1 \;\; C_2 \;\; C_3 \;\; C_4$$

Fig. 2: Crossbar $\mathcal{X} = (M, R = (R_1, R_2, R_3), C = (C_1, C_2, C_3, C_4))$ with well-formed design (6). If a voltage pulse is applied to R_1 and R_2, C_3, C_4 are grounded, then we have an initial flow of current $r_1 = 1$. The red bars represent the current flow from R_1 to C_3 when $y > x$, i.e. when $y \wedge \neg x$ holds.

We must make a distinction between the case where the input wire values are constant (i.e. $s_i = 1$ for all $\alpha \in \mathbb{B}^p$), as is the case in Fig. 2, and the case where the value of an input wire depends on the evaluation vector $\alpha \in \mathbb{B}^p$. It has been shown by Jha et al. that any Boolean formula can be computed on a constant-input homogeneous crossbar using paths-based logic [9]. As we demonstrate in Theorem 1, this is not the case under variable inputs. In particular, we show that ripple-carry addition cannot be computed in variable-input homogeneous crossbars.

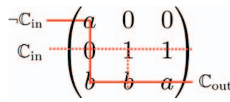
An adder is defined by its sum and carry-out bits $\mathbb{S}$ and $\mathbb{C}_{\text{out}}$. Given bits a, b, and carry-in values $\neg\mathbb{C}_{\text{in}}$ and $\mathbb{C}_{\text{in}}$, we define $(\mathbb{S}|\neg\mathbb{C}_{\text{in}}), (\mathbb{S}|\mathbb{C}_{\text{in}}), (\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}), (\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}})$ in (7), where $(\phi|\psi)$ is a formula denoting the value of ϕ when ψ holds.

$$\begin{aligned} \mathbb{S} =& (a \wedge \neg b \wedge \neg\mathbb{C}_{\text{in}}) \vee (\neg a \wedge b \wedge \neg\mathbb{C}_{\text{in}}) \vee \\ & (\neg a \wedge \neg b \wedge \mathbb{C}_{\text{in}}) \vee (a \wedge b \wedge \mathbb{C}_{\text{in}}) \\ \mathbb{C}_{\text{out}} =& (a \wedge b) \vee (a \wedge \mathbb{C}_{\text{in}}) \vee (b \wedge \mathbb{C}_{\text{in}}) \\ (\mathbb{S}|\neg\mathbb{C}_{\text{in}}) =& (a \wedge \neg b) \vee (\neg a \wedge b), (\mathbb{S}|\mathbb{C}_{\text{in}}) = (\neg a \wedge \neg b) \vee (a \wedge b) \\ (\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}) =& (a \wedge b), (\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}}) = a \vee b \end{aligned} \quad (7)$$

The following theorem may seem esoteric, but it provides a guideline for determining which functions cannot be computed on variable-input homogeneous crossbars using paths-based logic. We motivate this limitation with a simple example.

Suppose we are given a crossbar $\mathcal{X} = (M, R, C)$ with variable inputs $S = (R_1, R_2)$ and output $F = (R_3)$ such that $s_1 = \neg\mathbb{C}_{\text{in}}$, $s_2 = \mathbb{C}_{\text{in}}$, and we want to compute $f_1 \iff \mathbb{C}_{\text{out}}$.

Use the following figure as a reference. There must be a path $\Pi^{S_1 \to F_1}$ to compute $(\mathbb{C}_{\text{out}}|s_1) = (\mathbb{C}_{\text{out}}|\neg\mathbb{C}_{\text{in}}) = a \wedge b$ and two paths $\Pi^{S_2 \to F_1}$, $\Pi'^{S_2 \to F_1}$ to compute the two clauses in $(\mathbb{C}_{\text{out}}|s_2) = (\mathbb{C}_{\text{out}}|\mathbb{C}_{\text{in}}) = (a) \vee (b)$. Note that when $\alpha = (a = 1, b = 1, \mathbb{C}_{\text{in}} = 0)$, we have $s_1 = 1$ and $s_2 = 0$. Thus, path $\Pi^{S_1 \to F_1}$ will yield $f_1 = 1$ as denoted by the solid red lines in the figure. However, path $\Pi'^{S_2 \to F_1}$ will then yield $s_2 = 1$ as shown by the dotted lines, which is a contradiction since $s_2 = \mathbb{C}_{\text{in}}$. This is due to the symmetry of paths property (3) in homogeneous crossbars.



Theorem 1. *There exists a class of Boolean formulas that cannot be computed on variable-input homogeneous crossbars using paths-based logic.*

Proof. Suppose there is a well-formed design P for a crossbar $\mathcal{X} = (M, R, C)$ with source/destination wires $S, F \subset W$ for some $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and assume the following statements hold under some evaluation vector $\alpha \in \mathbb{B}^p$ for some $\beta, \omega, i, j, k, k'$: (i) $s_\beta \wedge \neg s_\omega$, (ii) ϕ^i is satisfied, (iii) $(\Phi^i_k|s_\omega) \subseteq (\Phi^i_{k'}|s_\beta)$, (iv) $(\phi^i_{k'}|s_\beta)$ is satisfied.

Since $(\Phi^i_k|s_\omega) \subseteq (\Phi^i_{k'}|s_\beta)$ and $(\phi^i_{k'}|s_\beta)$ is satisfied, $(\phi^i_k|s_\omega)$ is also. For every $S_h \in S$, there must exist a path $\Pi^{S_h \to F_{h'}}$ for each clause in $(\Phi^{h'}|s_h)$. Thus, there must be paths $\Pi^{S_\beta \to F_i}$, $\Pi^{S_\omega \to F_i}$ corresponding to $(\Phi^i_{k'}|s_\beta)$ and $(\Phi^i_k|s_\omega)$, respectively. Since we have chosen α such that $(\phi^i_{k'}|s_\beta)$, $(\phi^i_k|s_\omega)$, and s_β are satisfied, implication (8) follows from the symmetry of paths property, yielding the contradiction $s_\omega \wedge \neg s_\omega$. □

$$\left(s_\beta \wedge \bigwedge_d \pi_d^{S_\beta \to F_i} \implies f_i\right) \wedge \left(f_i \wedge \bigwedge_d \pi_d^{S_\omega \to F_i} \implies s_\omega\right) \quad (8)$$

Suppose we want to find a well-formed design P for a ripple-carry adder. Given a crossbar $\mathcal{X} = (M, R, C)$ with source/destination wires $S, F \subset W$, let the inputs be defined by $s_1 = \neg\mathbb{C}_{\text{in}}$ and $s_2 = \mathbb{C}_{\text{in}}$, where $(\mathbb{C}_{\text{out}}|s_1) = \{\{A, B\}\}$ and $(\mathbb{C}_{\text{out}}|s_2) = \{\{A\}, \{B\}\}$. Given evaluation vector $\alpha = (a = 1, b = 1, \mathbb{C}_{\text{in}} = 0)$, conditions (i)–(iv) from Theorem 1 hold. Indeed, we have (i) $s_1 \wedge \neg s_2$, (ii) $\mathbb{C}_{\text{out}}$ is satisfied, (iii) $(\mathbb{C}_{\text{out}_1}|s_2) \subseteq (\mathbb{C}_{\text{out}_1}|s_1)$, and (iv) $(\mathbb{C}_{\text{out}_1}|s_1)$ is satisfied. From Theorem 1, we know that symmetry (3) yields s_2. This is a contradiction since $s_2 = 0 = \mathbb{C}_{\text{in}}$ holds due to α.

IV. Heterogeneous Crossbars

The use of heterogeneous crossbar designs defined below gives us greater control over the paths induced by allowing the use of unidirectional components. This changes the dynamics of the crossbar. The flow behavior in heterogeneous crossbars follows Axiom 2. The mapping matrix P, where $P_{ij} \in \{B_1, \neg B_1, B_2, \neg B_2, \ldots, B_p, \neg B_p\} \cup \{0, 1, D\}$, satisfies (4) and $(m_{ij} = D) \iff (P_{ij} = D)$.

Definition 3. *A heterogeneous crossbar is a crossbar $\mathcal{X} = (M = (m_{ij}), R, C)$, where each m_{ij} is chosen from a set of multiple components. For the purposes of our paper, this would be the set $\{0, 1, D\}$ of bidirectional OFF and ON components and row-to-column unidirectional components, respectively.*

Axiom 2 (Heterogeneous Flow)**.** *Given a heterogeneous crossbar $\mathcal{X} = (M \in \{0, 1, D\}^{|R| \times |C|}, R, C)$, $(r_i \wedge m_{ij} \in \{1, D\}) \implies c_j$ and $(c_j \wedge m_{ij} = 1) \implies r_i$ hold. Consequently, equation (9) holds.*

$$\begin{aligned} &\left(\bigwedge_{j=1}^{|C|} c_j \iff \left(\bigvee_{i=1}^{|R|} (m_{ij} \in \{1, D\} \wedge r_i)\right)\right) \wedge \\ &\left(\bigwedge_{i=1}^{|R|} r_i \iff \left(\bigvee_{j=1}^{|C|} (m_{ij} = 1 \wedge c_j)\right)\right) \end{aligned} \quad (9)$$

It follows from (9) that symmetry (3) does not hold for a path $\Pi^{W_i \to W_j}$ with $D \in \pi^{W_i \to W_j}$. This is intuitive since D is a unidirectional component that allows the flow of current to traverse from rows to columns, but suppresses current in the opposite direction. That is, if $m_{ij} = D$, we have $(c_j \wedge m_{ij}) \not\Longrightarrow r_i$. This would violate the symmetry equation (3). Recall that symmetry causes the contradiction in Theorem 1.

V. Design Automation

Bounded model checking (BMC) algorithms construct a formula $\mathcal{M}_{\text{BMC}}$ (13) from a state-transition system based on an initialization condition I, a transition relation τ, a specification to be checked ψ, and a time bound T denoting the maximum length of a trajectory in the state-transition graph [10]. If a state is found wherein $\mathcal{M}_{\text{BMC}}$ is evaluated to be false, a counterexample is produced. We will build a state-transition system such that said counterexample produces a well-formed design P that computes a given formula ϕ.

Given $\phi : \mathbb{B}^p \mapsto \mathbb{B}^q$ and a design P for some variable-input crossbar $\mathcal{X} = (M, R, C)$ with source and destination wires $S, F \subset W$, we have 2^p evaluation vectors $\alpha \in \mathbb{B}^p$. For each such α, we define a finite state machine $\mathcal{L}_\alpha$, where $\mathcal{L}_\alpha[m_{ij}]$, $\mathcal{L}_\alpha[r_i]$, and $\mathcal{L}_\alpha[c_j]$ denote the values of m_{ij}, r_i, and c_j induced by P under evaluation α. Each $\mathcal{L}_\alpha$ is a component in a transition system $\mathcal{L}$ consisting of an initial state I (10) and a transition relation τ (11) defining the dynamics of the system as it moves from state u_t to state u_{t+1}. We are looking for a state u_t such that ψ (12) is violated. We have defined (12) as the negation of (5) so that a counterexample to (12) yields a well-formed design P, i.e. one where (5) holds.

In order to determine the value of bound T, it is important to know the results in [11], where Velasquez et al. prove that the set of paths of length at most $2(\min\{|R|, |C|\})$ between any two wires has the same computing power under paths-based logic as the set of all paths between said wires [11]. Using equation (11) as a reference, note that each transition from u_t to u_{t+1} corresponds to some M_{ij} redirecting current from one of its terminals to the other. Thus, we need only look at trajectories in $\mathcal{L}$ of length at most to $T = 2(\min\{|R|, |C|\})$ to determine whether a counterexample to (12) exists.

	[1]	[2]	[3]	[3]	[4]	[5]	[6]	[6]	[7]	[7]	XRCA
Execution Steps	$19n$	$8n+12$	$29n$	$5n+18$	$89n$	$15n$	$13n+2$	$7n+21$	$2n+4$	$4n+5$	7
Crossbar Nodes	$5n+1$	$35n$	$3n+3$	$9n$	$3n+5$	$5n+6$	$3n+6$	$8n$	$2n+2$	$n+2$	$30n$

TABLE I: Comparison of our crossbar ripple-carry adder (XRCA) against other crossbar n-bit adder designs proposed in the literature.

$$\begin{aligned} I(u_0) \triangleq \bigwedge_{\alpha \in \mathbb{B}^p} &\left(\bigwedge_{w_i \in S} (\mathcal{L}^0_\alpha[w_i] \iff s_i) \right) \wedge \left(\bigwedge_{w_i \notin S} \neg \mathcal{L}^0_\alpha[w_i] \right) \wedge \left(\bigwedge_{i,j} (\mathcal{L}_\alpha[m_{ij}] = D) \iff (P_{ij} = D) \right) \wedge \\ &\left(\bigwedge_{i,j,k} (\mathcal{L}_\alpha[m_{ij}] = 1) \iff (P_{ij} = 1) \vee ((P_{ij} = B_k) \wedge b_k) \vee ((P_{ij} = \neg B_k) \wedge \neg b_k) \right) \end{aligned} \tag{10}$$

$$\tau(u_t, u_{t+1}) \triangleq \bigwedge_{\alpha \in \mathbb{B}^p} \bigwedge_i \left(\mathcal{L}^{t+1}_\alpha[r_i] \iff \bigvee_j \mathcal{L}_\alpha[m_{ij}] = 1 \wedge \mathcal{L}^t_\alpha[c_j] \right) \wedge \bigwedge_j \left(\mathcal{L}^{t+1}_\alpha[c_j] \iff \bigvee_i \mathcal{L}_\alpha[m_{ij}] \in \{1, D\} \wedge \mathcal{L}^t_\alpha[r_i] \right) \tag{11}$$

$$\psi(u_t) \triangleq \neg \left(\bigwedge_{\alpha \in \mathbb{B}^p} \left(\bigwedge_{k=1}^{q} \left(\mathcal{L}^t_\alpha[f_k] \iff \phi^k \right) \right) \right) \tag{12}$$

$$\mathcal{M}_{\text{BMC}} \triangleq I(u_0) \wedge \bigwedge_{t=1}^{T-1} \tau(u_t, u_{t+1}) \wedge \bigwedge_{t=1}^{T} \psi(u_t) \tag{13}$$

$$\begin{gathered} (\mathbb{S}_i | \neg \mathbb{C}_{i-1}) = (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i) \\ (\mathbb{S}_i | \mathbb{C}_{i-1}) = (\neg x_i \wedge \neg y_i) \vee (x_i \wedge y_i) \\ (\mathbb{C}_i | \neg \mathbb{C}_{i-1}) = (x_i \wedge y_i), (\mathbb{C}_i | \mathbb{C}_{i-1}) = x_i \vee y_i \end{gathered} \tag{14}$$

By providing the model checking formula $\mathcal{M}_{\text{BMC}}$ (13) to the NuSMV 2.6.0 model checker, we synthesized a mapping for a k-bit ripple-carry adder based on equations (14) given two k-bit vectors $x, y \in \{0,1\}^k$. The well-formed design P^i for the crossbar $\mathcal{X}^i = (M^i, R^i, C^i)$ with source and destination wires $S = (R^i_1, R^i_2), F = (R^i_4, R^i_5, C^i_5)$ computes $(\neg \mathbb{C}_i, \mathbb{C}_i, \mathbb{S}_i)$, where $s^i_1 = \neg \mathbb{C}_{i-1}$ and $s^i_2 = \mathbb{C}_{i-1}$ are carry-in values. That is, P^i is a 1-bit full adder. The design can be seen in (15). Placing n of these in series yields an n-bit adder.

$$\begin{pmatrix} D & 0 & 0 & 0 & 0 \\ 0 & y_i & y_i & \neg y_i & 0 \\ D & 0 & x_i & \neg x_i & 1 \\ 1 & 0 & \neg y_i & y_i & 0 \\ \neg x_i & 0 & \neg y_i & 0 & 0 \\ 0 & \neg x_i & 0 & x_i & 0 \end{pmatrix} \tag{15}$$

It is known that a crossbar $\mathcal{X} = (M, R, C)$ can be configured in $\min\{|R|, |C|\} + 1$ steps [12]. Thus, each crossbar in our n-bit adder can be programmed independently in 6 steps, with a read voltage then applied to read the output wire values. Therefore, we can compute n-bit addition using a constant number of steps. See Table I for a comparison with other approaches.

VI. Experimental Results

We utilize HSPICE for our experiments. For each $m_{ij} = 1$ ($m_{ij} = 0$), we use resistors with LRS (HRS) resistance $R_{LRS} = 10\Omega$ ($R_{HRS} = 1M\Omega$) and we implement the SDM02U30CSP diode model from Diodes Incorporated® [13] for unidirectional components $m_{ij} = D$. Resistors-to-ground with resistance $R_G = 500\Omega$ are placed before each grounded wire. See Table II for simulation results of design (15).

	000	001	010	011	100	101	110	111
C_5	$17.4m$	4.544	4.627	$12.4m$	4.717	$14.7m$	$22.1m$	4.551
R_5	4.902	4.567	4.718	$12.5m$	4.807	$14.7m$	$4.97m$	$11.7m$
R_6	$9.89m$	$18.6m$	$24.4m$	4.807	$28.9m$	4.807	4.807	4.38

TABLE II: HSPICE simulation results for the full adder design (15). Each column entry denotes values under an evaluation vector $\alpha = (x_i, y_i, \mathbb{C}_{i-1})$. Wires C_5, R_5, R_6 are grounded and a $5V$ voltage pulse is applied to R_1 if $\mathbb{C}_{i-1} = 0$ or to R_2 if $\mathbb{C}_{i-1} = 1$. The voltage values read correspond to $\mathbb{S}_i, \neg \mathbb{C}_i, \mathbb{C}_i$, respectively. Each entry denotes the voltage reading obtained from wires C_5, R_5, R_6.

References

[1] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magicmemristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[2] Divya Mahajan, Matheen Musaddiq, and Earl E Swartzlander. Memristor based adders. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, pages 1256–1260. IEEE, 2014.

[3] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Memristor-based material implication (imply) logic: Design principles and methodologies. 2013.

[4] Eero Lehtonen and Mika Laiho. Stateful implication logic with memristors. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 33–36. IEEE Computer Society, 2009.

[5] Eero Lehtonen, Jussi Poikonen, and Mika Laiho. Implication logic synthesis methods for memristors. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2441–2444. IEEE, 2012.

[6] Yuanfan Yang, Jimson Mathew, Salvatore Pontarelli, Marco Ottavi, and Dhiraj K Pradhan. Complementary resistive switch-based arithmetic logic implementations using material implication. *IEEE Transactions on Nanotechnology*, 15(1):94–108, 2016.

[7] Anne Siemon, Stephan Menzel, Rainer Waser, and Eike Linn. A complementary resistive switch-based crossbar array adder. *IEEE journal on emerging and selected topics in circuits and systems*, 5(1):64–74, 2015.

[8] P Kuekes. Material implication: digital logic with memristors. In *Memristor and memristive systems symposium*, volume 21, 2008.

[9] Sumit Kumar Jha, Dilia E Rodriguez, Joseph E Van Nostrand, and Alvaro Velasquez. Computation of boolean formulas using sneak paths in crossbar computing, April 19 2016. US Patent 9,319,047.

[10] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 19(1):7–34, 2001.

[11] Alvaro Velasquez and Sumit Kumar Jha. Fault-tolerant in-memory crossbar computing using quantified constraint solving. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 101–108. IEEE, 2015.

[12] A. Velasquez and S. K. Jha. Parallel boolean matrix multiplication in linear time using rectifying memristors. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1874–1877, May 2016.

[13] Diodes Incorporated. Spice models. http://www.diodes.com/spicemodels/search.php. Accessed: 2016/12/04.