Computation of Boolean Formulas Using Sneak Paths in Crossbar Computing

2014

Alvaro Velasquez
University of Central Florida

COMPUTATION OF BOOLEAN FORMULAS USING SNEAK PATHS IN CROSSBAR COMPUTING

by

ALVARO VELASQUEZ

A thesis submitted in partial fulfilment of the requirements
for the degree of Honors in the Major, B.S.
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida

Spring Term
2014

Advisor: Sumit Kumar Jha

ABSTRACT

Memristor-based nano-crossbar computing is a revolutionary computing paradigm that does away with the traditional Von Neumann architectural separation of memory and computation units. The computation of Boolean formulas using memristor circuits has been a subject of several recent investigations. Crossbar computing, in general, has also been a topic of active interest, but sneak paths have posed a hurdle in the design of pervasive general-purpose crossbar computing paradigms. In this paper, we demonstrate that sneak paths in nano-crossbar computing can be exploited to design a Boolean-formula evaluation strategy. We demonstrate our approach on a simple Boolean formula and a 1-bit addition circuit. We also conjecture that our nano-crossbar design will be an effective approach for synthesizing high-performance customized arithmetic and logic circuits.

Contents

Chapter 1: Introduction

In 1971, Leon Chua postulated the existence of a new circuit element. The three basic two-terminal circuit elements then known connect pairs of the four circuit variables: current i, voltage v, charge q, and magnetic flux ϕ. Five relationships combining these variables were known: the definition of current, the definition of voltage from the induction law of Faraday, and the three axiomatic definitions of the known circuit elements. Completeness required, Chua argued, a sixth relationship, which would define the missing circuit element, see Figure 1. He called this missing element the *memristor*, a contraction of memory and resistor, for its behavior was somewhat like that of a nonlinear resistor with memory. Not until 2008 was it physically realized by Stanley Williams and his team at HP Labs [8].

Since then, much research has sought to exploit its properties, from a quest for memristive memories, to memristive devices for neuromorphic computing, to computational logic that uses memristors as logic gates. The focus of this paper is the evaluation of Boolean formulas, which has garnered some recent attention, and is an enabling technology for synthesizing high-performance, low-power arithmetic and logic circuits.

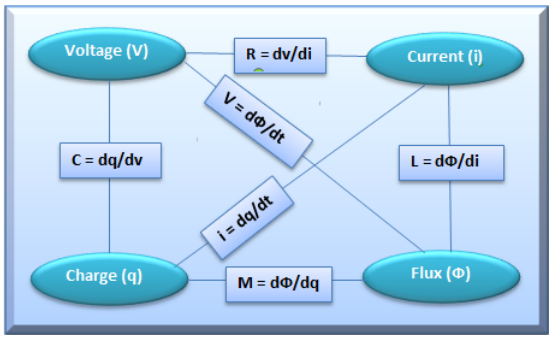


Figure 1.1: Relations between different elements and physical properties of circuits.

Space limitations preclude an exhaustive survey of the literature on memristive evaluation of Boolean

formulas. We consider briefly some representative work in this area. Borghetti, Snider *et al.* use two memristors to compute material implication, a fundamental operation in Boolean logic [2]. They leverage this design to synthesize the universal nand gate using three memristors and three time steps, one initialization step and two implication steps. Gale, de Lacy Costello *et al.* use a single memristor, through which two pulses pass sequentially, with the second being sent before the current spike generated by the first stabilizes [4]. The logical value true is obtained when the current spike goes past a threshold. These designs have two fundamental problems. First, they use a few memristors interacting with other non-memristive circuit elements. It is difficult to put such a heterogeneous mixture of circuit elements on the same hybrid chip. Second, they rely on a sequence of correctly timed inputs, requiring a global clock. Such synchronous extreme-scale circuits are notoriously difficult to design.

For purposes of fabrication, rather than using a few individual memristors, it is more natural to organize sets of them into crossbar networks. However, if the design or application requires that memristors be addressed individually, then the *sneak-path problem* arises: current flows through unknown paths in parallel to the memristor, which prevents a correct detection of its resistance or flow of current through it. Much research is being devoted to overcoming this problem. The proposed solutions often unavoidably increase the complexity of the fabrication process.

The evaluation of Boolean formulas forms the base of much of computing, including logical and arithmetic calculations. We propose an efficient and low-energy method that exploits asynchronous memristor-based crossbars circuits that use sneak paths as a first-class design primitive. In Section 2 we present a brief background on crossbar computing, and define formal concepts to describe it, which we use in Section 3 to present a crossbar design for the evaluation of Boolean formulas in negation normal form. Section 4 describes experiments in which this evaluation approach is applied to arithmetic circuits for addition. Concluding remarks follow in Section 5.

Chapter 2: Crossbar Computing

Memristors

The stipulated memristor furnished the missing functional relation among the circuit variables, relating magnetic flux ϕ and charge q: $d\phi = M dq$. Since $d\phi = v\,dt$ and $dq = i\,dt$, the function M defining the memristor may be expressed equivalently as $v = Mi$, which when linear is identical with resistance R, but when nonlinear and a function of the state of the memristor is a new property. The HP Labs team developed a nanoscale *current-controlled memristor*, which is defined by:

$$v = R(w)\, i \qquad \frac{dw}{dt} = i$$

where $R(w)$ is a generalized resistance that depends on an internal state variable w that is proportional to charge. The memristor consists of a thin semiconductor film of thickness d between two metal contacts. The film has two regions: a doped region, with a high concentration of dopants, and an undoped region, with a nearly zero concentration of dopants. Applying an external bias $v(t)$ causes the charged dopants to drift, and the boundary between the regions moves. A memristor with a doped region of length d has low resistance, R_{on}; while one with an undoped region of length d has the much higher resistance R_{off}. A memristor can be viewed as two variable resistors connected in series [8][9]:

$$v(t) = \left[\frac{w(t)}{d} R_{on} + \left(1 - \frac{w(t)}{d}\right) R_{off}\right] i(t)$$

where the charge-dependent state variable $w(t)$ is the width of the doped region of the memristor. For the case of ohmic electronic conduction and linear ionic drift in a uniform field with average ion mobility μ, [8] defines

$$w(t) = \mu \frac{R_{on}}{d} q(t).$$

For $R_{on} \ll R_{off}$, a simple analysis leads to the first definition of memristance in terms of the material and geometrical properties of the memristor:

$$M(q) \equiv \frac{d\phi}{dq} = R_{off}\left(1 - \frac{\mu R_{on}}{d^2} q\right).$$

We define a memristor by its physical parameters as follows:

Definition 1 MEMRISTOR *A memristor is a 4-tuple $M \equiv (R_{off}, R_{on}, d, \mu)$, where R_{off} is the resistance of the fully turned-off memristor, R_{on} is the resistance of the fully turned-on memristor, with $R_{on} \ll R_{off}$, and d and μ are the width and the average ionic mobility of the memristor.*

Note that the q-dependent term in the definition of the memristor, which makes its memristance nonlinear, becomes larger as the memristor width d becomes smaller. Thus, this term becomes more critical in understanding the behavior of electronic devices as they shrink to the nanoscale. Also, at this scale, small voltages can yield very large electric fields, which can produce nonlinearities in ionic transport. These affect the rate of change of state variable w as it approaches boundaries 0 and d. This phenomenon is called the *nonlinear dopant drift*.

Various *window functions* f have been proposed to model it [8, 1, 5, 7, 6]. Strukov *et al.* proposed the first: $f(w) = w(d-w)/d^2$ [8]. Letting $x = w/d \in [0,1]$, it can be rewritten as $f(x) = x - x^2$. More recently, Prodomakis *et al.* have proposed $f(x) = 1-[(x-0.5)^2+0.75]^p$, where p is a control parameter that can take any positive real number. It is scalable, which means that $0 \leq f_{max}(x) \leq 1$, and can be adjusted to fit experimental observations.

We define the dynamics of a memristor as follows:

Definition 2 DYNAMICAL MODEL OF A MEMRISTOR *The evolution of a memristor M is described by a 3-tuple $D(M) \equiv (f, v, w)$, where f is a window function, $v(t)$ is the voltage applied across the two-terminal memristor at time t, and $w(t)$ is the width of the doped region of memristor M at time t.*

Taking into account the nonlinear dopant drift, the width of the doped region of the memristor changes as follows:

$$\delta w(t) = \mu \frac{R_{on}}{d} \left[\frac{v(t)\delta t}{\frac{w(t)}{d} R_{on} + \left(1 - \frac{w(t)}{d}\right) R_{off}} \right] f\left(\frac{w(t)}{d}\right).$$

We exploit the behavior of memristors to define the state space that will allow us to compute with memristors. Assuming $R_{on} \ll R_{off}$, a sequence of resistance values or *digitization schedule*: $R_{on} \equiv R_0 < \cdots < R_{n-2} < R_{off} \equiv R_{n-1}$, for $n \geq 2$, serves to define a corresponding set of states. Let $\Delta R_i = R_{i+1} - R_i$, for $0 < i \leq n-1$. Each state is a range of resistance values defined as follows:

$$s_0 = \left[R_0, R_0 + \frac{\Delta R_1}{2}\right] \qquad s_{n-1} = \left[R_{n-2} + \frac{\Delta R_{n-1}}{2}, R_{n-2}\right]$$

$$s_i = \left[R_{i-1} + \frac{\Delta R_i}{2}, R_i + \frac{R_{i+1}}{2}\right], \text{ for } 0 < i < n-1.$$

In this paper, we focus on memristors with three states: ON, OFF, UNDECIDED. The ON state will be used to guide the current through the sneak path, while the OFF state will be used to prevent current from flowing through multiple paths. The UNDECIDED state will be used to robustly separate the ON state from the OFF state. In the rest of the paper, we do not belabor the separation of the ON and OFF states by the UNDECIDED state, but we note here that this separation is important for robust evaluation of Boolean formulas.

Crossbars

It is natural to organize sets of memristors into crossbar networks: a top set of parallel nanowire electrodes, a bottom set of parallel nanowire electrodes perpendicular to the top one, and a memristor at each crosspoint, see Figure 2. The memristor models presented above induce corresponding models for crossbars. They lay the foundation for our approach to crossbar computing.

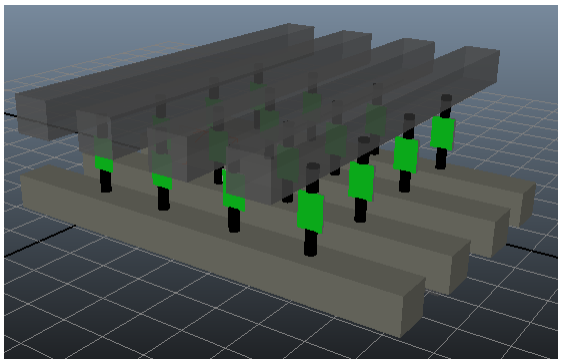


Figure 2.1: Memristor crossbar.

The physical description of a crossbar is given by:

Definition 3 CROSSBAR *A (memristor-based) crossbar is a 3-tuple* $\mathbb{C} = (\mathbb{M}, \mathbb{X}, \mathbb{Y})$ *where*

1. $\mathbb{M} = \begin{pmatrix} M_{m,1} & M_{1,2} & \dots & M_{m,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{1,1} & M_{m,2} & \dots & M_{1,n} \end{pmatrix}$ *is a two-dimensional array of memristors with* m *rows and* n *columns;*

2. $\mathbb{X} = (X_1, \dots, X_n)$ *is the vector of X-crossbars (or nanowires); crossbar* X_j *provides the same input to every memristor in column* j*.*

3. $\mathbb{Y} = (Y_1, \dots, Y_m)$ *is the vector of Y-crossbars (or nanowires); crossbar* Y_i *provides the same input to every memristor in row* i*.*

We make explicit the size of a matrix $\mathbb{M}$ with m rows and n columns by writing $\mathbb{M}^{m,n}$. Similarly, to make explicit the length of a vector $\mathbb{V}$, we write $\mathbb{V}^m$, or either $\mathbb{V}^{m,1}$ or $\mathbb{V}^{1,m}$.

The dynamics of a crossbar are defined by the following model:

Definition 4 DYNAMICAL MODEL OF A CROSSBAR *Given a crossbar* $\mathbb{C} \equiv (\mathbb{M}^{m,n}, \mathbb{X}^n, \mathbb{Y}^m)$*. The evolution of* $\mathbb{C}$ *is described by the 4-tuple* $D(\mathbb{C}) \equiv (\mathbb{F}^{m,n}, \mathbb{V}^n_{\mathbb{X}}, \mathbb{V}^m_{\mathbb{Y}}, \mathbb{W}^{m,n})$*, where*

1. *$\mathbb{F}^{m,n}$ consists of the window functions for the memristors in $\mathbb{M}^{m,n}$, such that $\mathbb{F}_{i,j}$ is the window function for memristor $\mathbb{M}_{i,j}$;*

2. *$\mathbb{V}^n_{\mathbb{X}}(t)$ consists of the voltages that are applied to the X-crossbars, such that voltage $\mathbb{V}_{\mathbb{X}i}$ is applied to crossbar X_i at time t;*

3. *$\mathbb{V}^m_{\mathbb{Y}}(t)$ consists of the voltages that are applied to the Y-crossbars, such that voltage $\mathbb{V}_{\mathbb{Y}i}$ is applied to crossbar Y_i at time t; and*

4. *$\mathbb{W}^{m,n}(t)$ consists of widths of the doped regions of the memristors in $\mathbb{M}^{m,n}$ at time t, which depend on the difference of the voltages applied to the X- and Y-crossbars at time t. Without loss of generality, the width of $\mathbb{M}_{i,j}$ at time t is given by:*

$$\delta\mathbb{W}_{i,j}(t) = \mu\frac{R_{on}}{d}\left[\frac{(\mathbb{V}_{\mathbb{Y}_i}(t) - \mathbb{V}_{\mathbb{X}_j}(t))\delta t}{\frac{\mathbb{W}_{i,j}(t)}{d}R_{on} + \left(1-\frac{\mathbb{W}_{i,j}(t)}{d}\right)R_{off}}\right] f\left(\frac{\mathbb{W}_{i,j}(t)}{d}\right)$$

Sneak Paths: A Problem

Sneak paths are interconnected nanowires through which current flows in crossbars, which in some applications are undesirable. They pose a problem when it is necessary to determine the resistance or the current flowing through an individual memristor. If this memristor is in the high-resistance state, sneak paths provide alternate paths for current to flow between the electrodes the memristor joins. In these applications, they are determined by often unknown series of low-resistance memristors. Thus, they act as unknown resistances in parallel with one of the memristors of interest. The result is that the memristor in high-resistance state appears erroneously to be in the low-resistance state.

A sneak path is a path that connects the row- and column-electrodes of a memristor in high-resistance state, and whose remaining segments are determined by memristors in low-resistance states [3]: There is a sneak path of length $2(k+1)$ relative to position (i,j) in memory array $\mathbb{M}^{m,n}$ if $\mathbb{M}_{i,j}$ is in a state with resistance R_{off} and there exist $2k$ positive integers $1 \leq r_1, \ldots, r_k \leq m$ and

$1 \leq c_1, \ldots, c_k \leq n$ such that memristors $\mathbb{M}_{i,c_1}$, $\mathbb{M}_{r_1,c_1}$, $\mathbb{M}_{r_1,c_2}$, $\ldots$, $\mathbb{M}_{r_{i-1},c_i}$, $\mathbb{M}_{r_i,c_i}$, $\mathbb{M}_{r_i,c_{i+1}}$, $\ldots$, $\mathbb{M}_{r_{k-1},c_k}$, $\mathbb{M}_{r_k,c_k}$, $\mathbb{M}_{r_k,j}$ are in a state with resistance R_{on}. Considering only the physical characteristics of such a path, a sneak path is one through which a significant amount of current flows. In the next chapter we design crossbars that induce a slight variation of these paths to evaluate Boolean expressions.

Chapter 3: Crossbar Design For Evaluation of Boolean Formulas

This chapter describes how some crossbar designs very naturally encode Boolean formulas, and how to use sneak paths to compute their value.

NNF: Negation Normal Form

An n-ary Boolean function maps an n-tuple of Boolean values to a Boolean value. It can be defined by a truth table of 2^n rows, one for each possible value an n-tuple may take. Alternatively and more concisely, it can be defined in terms of a few Boolean operators or connectives. A set of Boolean connectives is complete if every Boolean function can be defined by an expression that uses only the connectives in that set. The set consisting of the $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction) connectives is complete. So any Boolean function can be defined by a well-formed formula (wff) constructed as follows: (i) a Boolean (propositional) variable p is a wff; (ii) if ϕ is a wff, $\neg\phi$ is a wff; (iii) if ϕ_1 and ϕ_2 are wffs, $\phi_1 \wedge \phi_2$ is a wff, and (iv) if ϕ_1 and ϕ_2 are wffs, $\phi_1 \vee \phi_2$ a wff.

Our method to evaluate Boolean formulas requires the formulas to be in negation normal form (NFF), which is defined by a slight change to these rules. The negation connective may not be applied to an arbitrary wff, but only to a propositional variable. A formula is in negation normal form if it is constructed as follows: (i) a literal, that is, a propositional variable p or $\neg p$, is in NNF, (ii) if ϕ_1 and ϕ_2 are in NNF, $\phi_1 \wedge \phi_2$ is in NNF, and (iii) if ϕ_1 and ϕ_2 are in NNF, $\phi_1 \vee \phi_2$ is in NNF.

Any wff constructed using only connectives $\neg$, $\wedge$ and $\vee$ can be transformed into an equivalent formula in negation normal form by repeatedly applying the De Morgan Laws: $\neg(p \wedge q) \equiv \neg p \vee \neg q$, and $\neg(p \vee q) \equiv \neg p \wedge \neg q$, and simplifying $\neg\neg p$ to p. Thus, the method presented below can be used to evaluate any Boolean function.

Sneak Paths: A Tool

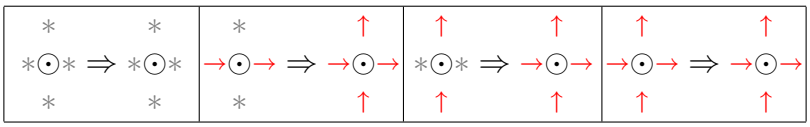
Some crossbar designs very naturally can encode a Boolean formula in negation normal form. The next section will describe them, and how to direct the flow of current so as to indicate the value of the formula. Sneak paths are nanowires connected by low-resistance memristors through which significant current flows. They are undesirable in applications such as memristor-based memories because the actual paths and currents through them are unknown, and the path of interest is one through a single memristor. The flow of current used to evaluate a Boolean formula is determined physically in the same way, but it is the flow of interest, and by design each segment of the path advances the evaluation of the formula. This section presents some basic ideas useful in describing the evaluation method.

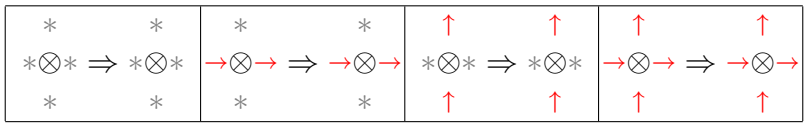
The state of a crossbar depends on the states of its memristors, and its X- and Y-crossbars. We introduce abstract models for these, which help us to reason formally about our paths of interest: those determined by memristors in low-resistance states.

As Figure 2 shows, a memristor in a crossbar joins two nanowires that are in different planes, and perpendicular to each other. In our abstraction, a nanowire (X- or Y-crossbar) may be in one of two states: *flow* and *no-flow*, which indicate whether current is flowing through it. Abstractly, a memristor is an operator that may change the states of the nanowires it joins. A memristor in a high-resistance state, denoted M_{off}, allows no current through it. The states of its nanowires cannot change, and thus M_{off} acts as an identity operator. A memristor in a low-resistance state, denoted M_{on}, allows current to flow through it. If its two nanowires are in the same state, the M_{on} operator preserves their states. When they are in different states, M_{on} takes the nanowire in *no-flow* state to the *flow* state.

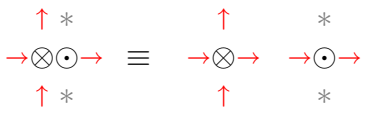
Graphically, let $\odot$ represent M_{on}, and $\otimes$ represent M_{off}, and let $\rightarrow$ represent the *flow* state of a nanowire, and $*$ represent the *no-flow* state. A 2-dimensional abstract view of a memristor and the nanowires it joins has the memristor in the center, the state of one nanowire above and below it,

and the state of the other to its left and right. Then the definitions of the memristor operations $\odot$ and $\otimes$ become:



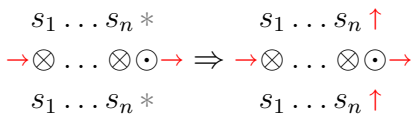


The physical sharing of nanowires in a row and in a column of a crossbar leads naturally to horizontal and vertical composition of memristor operations. We adopt a simpler notation to denote the state of a segment of a row or column of a crossbar. For example, for neighboring memristors in states M_{off} and M_{on}, we write:

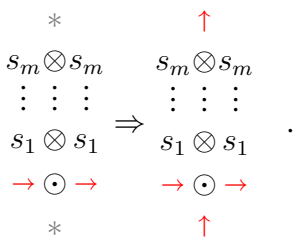


Sneak paths are determined by a set of M_{on} memristors. Consider now memristor-row and -column segments that are operations to construct sneak paths. Let a subscripted s denote the state of a nanowire. It follows from the definitions of the $\otimes$ and $\odot$ operations that current flowing through a row may be directed to a column without current by the following operations:

r2c.r



r2c.c



.

Similarly, current flowing through a column may be directed to a row without current by the following operations:

c2r.r

$$\begin{array}{c} \uparrow s_1 \ldots s_n \\ * \odot \otimes \ldots \otimes * \\ \uparrow s_1 \ldots s_n \end{array} \Rightarrow \begin{array}{c} \uparrow s_1 \ldots s_n \\ \rightarrow \odot \otimes \ldots \otimes \rightarrow \\ \uparrow s_1 \ldots s_n \end{array}$$

c2r.c

$$\begin{array}{c} \uparrow \\ * \odot * \\ s_m \otimes s_m \\ \vdots \ \vdots \ \vdots \\ s_1 \otimes s_1 \\ \uparrow \end{array} \Rightarrow \begin{array}{c} \uparrow \\ \rightarrow \odot \rightarrow \\ s_m \otimes s_m \\ \vdots \ \vdots \ \vdots \\ s_1 \otimes s_1 \\ \uparrow \end{array} .$$

As with operations $\odot$ and $\otimes$, these operations never take a nanowire from state $\rightarrow$ to state $*$. Below we will omit the s states for greater visual simplicity.

One other useful operation in this abstraction is an array of memristors in state $\otimes$: it is an identity operation on the set of nanowires it intersects. We will denote an m by n array by $\otimes^{m,n}$.

These operations serve to construct sneak paths, and it is a sneak path that is used to evaluate a Boolean formula.

Crossbar Design to Evaluate NNFs

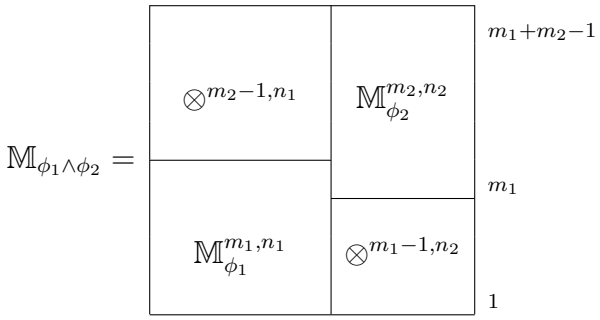
Every Boolean function can be defined by a Boolean formula in negation normal form. This section specifies crossbar states that can be used to evaluate such formulas using sneak paths.

Let ϕ be a Boolean formula in negation normal form. We define an array of memristors $\mathbb{M}_\phi$ as follows.

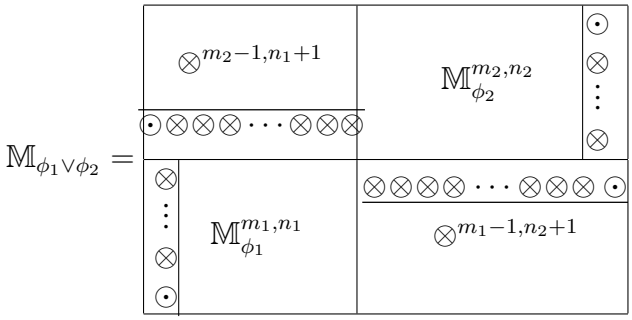
l: For $\phi = l$, a literal, let $\bigcirc = \odot$ if l is true, and $\bigcirc = \otimes$ if false. Then

$$\mathbb{M}_l = \begin{array}{c} \odot \\ \bigcirc \end{array}$$

c: For $\phi = \phi_1 \wedge \phi_2$ the array has $m_1 + m_2 - 1$ rows and $n_1 + n_2$ columns, and is defined by:



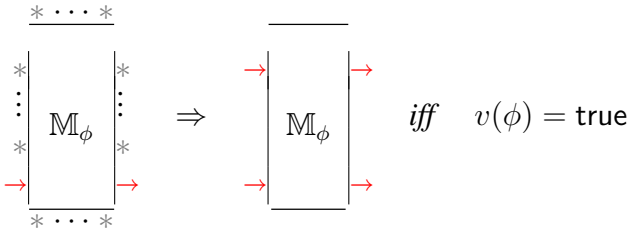
d: For $\phi = \phi_1 \vee \phi_2$ the array has $m_1 + m_2$ rows and $n_1 + n_2 + 2$ columns, and is defined by:



The memristor array $\mathbb{M}_\phi$ is an operator on the nanowires it intersects. To compute the value of ϕ, denoted $v(\phi)$, it must be applied to a set of nanowires in which only the first (or bottom) row is in state $\rightarrow$, and all the others are in the $*$.

We prove that given a crossbar in the state just described, which is induced by the formula ϕ, there exists a sneak path that takes current from the first (or bottom) row to the last (or top) row if and only if the value of ϕ is true.

Theorem 1 CROSSBAR COMPUTATION OF NNFS



Proof. The proof is by structural induction on the negation normal form ϕ. We prove that given a crossbar state in which only the nanowire in the first row is in state $\rightarrow$, a memristor array $\mathbb{M}_\phi$ creates a sneak path that takes the nanowire in the last row to state $\rightarrow$, if and only if $v(\phi) = \mathsf{true}$.

Base case: literal. $\phi \equiv l$. Assume the row 1 nanowire is in state $\rightarrow$, and the rest in state $*$. (i) For $v(l) = \mathsf{true}$, the $\odot$ in row 1 takes (the nanowire in) column 1 to state $\uparrow$, and the $\odot$ in row 2 takes row 2 to state $\rightarrow$. (ii) For $v(l) = \mathsf{false}$, the $\otimes$ in row 1 preserves the $*$ state of column 1, and the $\odot$ in row 2 preserves the $*$ state of row 2.

Inductive step: conjunction. $\phi \equiv \phi_1 \wedge \phi_2$. Assume row 1 is in state $\rightarrow$, and the rest of the nanowires are in state $*$. The required initial state for the nanowires of $\mathbb{M}_{\phi_1}$ follows immediately. For $\mathbb{M}_{\phi_2}$, note that the only nanowire in common with $\mathbb{M}_{\phi_1}$ is row m_1. For the rest of its nanowires, their initial $*$ state is preserved by the $\otimes$ arrays. Then by the induction hypothesis, $\mathbb{M}_{\phi_1}$ creates a sneak path that takes row m_1 to state $\rightarrow$ iff $v(\phi_1) = \mathsf{true}$. (i) For $v(\phi_1) = \mathsf{false}$ there is no sneak path that takes row m_1 to state $\rightarrow$. So the required initial state for $\mathbb{M}_{\phi_2}$ is not reached, and no sneak path to the last row of $\mathbb{M}_{\phi_1 \wedge \phi_2}$ can be created. (ii) For $v(\phi_1) = \mathsf{true}$ there is a sneak path that takes row m_1 to state $\rightarrow$. Then by the induction hypothesis, $\mathbb{M}_{\phi_2}$ creates a sneak path that takes its last row to state $\rightarrow$ iff $v(\phi_2) = \mathsf{true}$. Thus, $\mathbb{M}_{\phi_1 \wedge \phi_2}$ creates a sneak path that takes its last row to state $\rightarrow$ iff $v(\phi_1) = \mathsf{true}$ and $v(\phi_2) = \mathsf{true}$.

Inductive step: disjunction. $\phi \equiv \phi_1 \vee \phi_2$. Assume row 1 is in state $\rightarrow$, and the rest of the nanowires are in state $*$. (i) We establish that the required state for the nanowires of $\mathbb{M}_{\phi_1}$ holds. The column segment to the left of $\mathbb{M}_{\phi_1}$ preserves the initial state of its rows: $\odot$ preserves state $\rightarrow$, and $\otimes$ preserves state $*$. Then by the induction hypothesis, $\mathbb{M}_{\phi_1}$ creates a sneak path that takes its last row to state $\rightarrow$ iff $v(\phi_1) = \mathsf{true}$. By r2c.r, the row segment to the right of $\mathbb{M}_{\phi_1}$'s last row takes the last column of $\mathbb{M}_{\phi_1 \vee \phi_2}$ to state $\uparrow$. Last, by c2r.c, the segment of this column above row m_1 takes the last row of $\mathbb{M}_{\phi_1 \vee \phi_2}$ to state $\rightarrow$. (ii) We establish that the required state for the nanowires of $\mathbb{M}_{\phi_2}$ holds. By r2c.c, the column segment to the left of $\mathbb{M}_{\phi_1}$ takes column 1 to state $\uparrow$. By c2r.r, the row segment to the left of the first row of $\mathbb{M}_{\phi_2}$ takes that row to state $\rightarrow$. The $\otimes$ arrays preserve

the $*$ state of all the other nanowires. Then by the induction hypothesis, $\mathbb{M}_{\phi_2}$ creates a sneak path that takes the last row of $\mathbb{M}_{\phi_1 \vee \phi_2}$ to state $\rightarrow$ iff $v(\phi_2) = \mathsf{true}$. Thus, $\mathbb{M}_{\phi_1 \vee \phi_2}$ creates a sneak path that takes its last row to state $\rightarrow$ iff $v(\phi_1) = \mathsf{true}$ or $v(\phi_2) = \mathsf{true}$.

Chapter 4: Experiments

We demonstrate our design on a small Boolean formula and a 1-bit addition circuit. Our experiments are illustrative. We sketch the circuit designs, and verify that they function correctly.

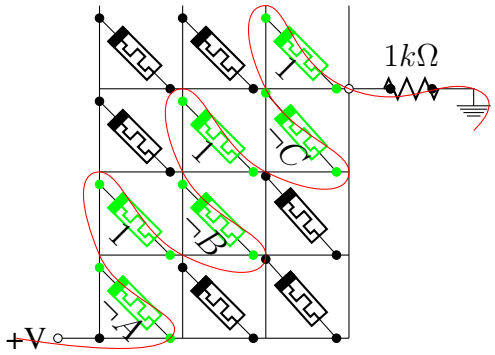


Figure 4.1: The design of a memristor based nano-crossbar for computing the formula $\neg A \wedge \neg B \wedge \neg C$.

Example Boolean formula

We built a memristor-based crossbar model for the formula $\neg A \wedge \neg B \wedge \neg C$ recognizing the pattern "A=0, B=0, C=0", in which we used 10,000 Ω as the turned-off resistance, and 1,000 Ω as the turned-on resistance. Figure 3 shows the turned-on memristors green, and the turned-off memristors black. The red curve shows the flow of the current through the only low-resistance path in the crossbar when the memristors for $\neg A$, $\neg B$, and $\neg C$ are turned-on. When any of these memristors is turned-off, there is no low-resistance path from $+V$ to ground in the crossbar.

Figure 4.2 shows the results of simulating this circuit using NGSPICE for two cases. For the only case in which the formula is true, when $\neg A$, $\neg B$ and $\neg C$ are true, the current flowing through the voltage source is of the order of 10^{-4} amperes. For the case when $\neg A$, $\neg B$ and $\neg C$ are false, and hence the formula is false, the current flowing through the voltage source is less than 10^{-5}.

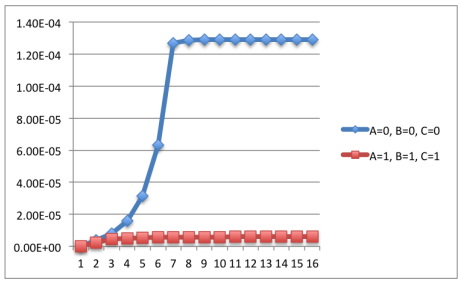


Figure 4.2: Current through the voltage source for the crossbar $\neg A \wedge \neg B \wedge \neg C$ under two different inputs.

1-bit addition circuit

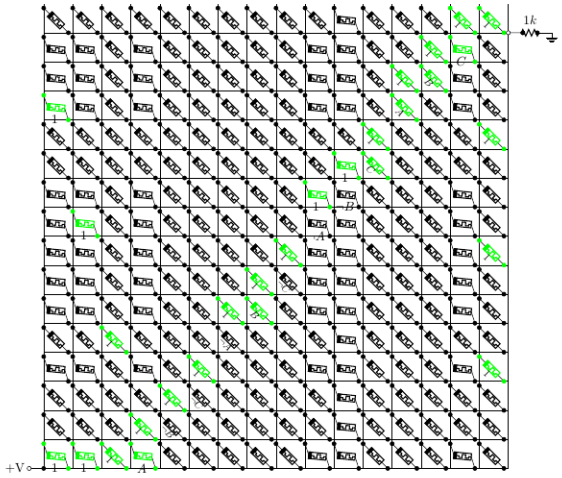


Figure 4.3: The design of a nano-crossbar for computing the sum-bit of a 1-bit adder whose Boolean formula is $(A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$

The nano-crossbar design for the 1-bit adder is shown in Figure 5, and the results of its simulation are presented in Table 1. Note that the measured voltage drop relates very well with the expected logical sum of the 1-bit adder: a voltage drop in excess of 1 volt indicates the logical value true, while a voltage drop below 0.5 volts indicates the logical value false.

A	B	C_{in}	Sum-Logical	Sum-Voltage
0	0	0	0	0.35
0	1	0	1	1.14
1	0	0	1	1.2
1	1	0	0	0.36
0	0	1	1	1.21
0	1	1	0	0.36
1	0	1	0	0.36
1	1	1	1	1.36

Table 4.1: Peformance of a nano-crossbar implementation of the 1-bit adder.

Chapter 5: Conclusion and Future Work

We have proposed a design for the evaluation of Boolean formulas using memristor-based crossbars. Our approach is counterintuitive. It transfigures sneak paths, well-known as a problem, into first-class design elements.

We will leverage this design to create a new algorithmic framework for constructing memristor-based nano-crossbar circuits that can implement programs involving arithmetic and logical operations as well as randomized algorithms. The resulting computing architecture will have applications in several important areas of computational data science and cyber-security, including extreme-scale simulation of complex systems such as agent-based models, biochemical reactions and fluid dynamics computations. By accelerating satisfiability solving, this research will also create new avenues for accelerating other NP-hard problems.

Bibliography

[1] Zdeněk Biolek, Dalibor Biolek, and Viera Biolkova. Spice model of memristor with nonlinear dopant drift. *Radioengineering*, 18(2):210–214, 2009.

[2] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. 'memristive'switches enable 'stateful'logic operations via material implication. *Nature*, 464(7290):873–876, 2010.

[3] Y. Cassuto, S. Kvatinsky, and E. Yaakobi. Sneak-path constraints in memristor crossbar arrays. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 156–160, 2013.

[4] Ella Gale, Ben de Lacy Costello, and Andrew Adamatzky. Boolean logic gates from a single memristor via low-level sequential logic. In *Unconventional Computation and Natural Computation*, pages 79–89. Springer, 2013.

[5] Yogesh N Joglekar and Stephen J Wolf. The elusive memristor: properties of basic electrical circuits. *European Journal of Physics*, 30(4):661, 2009.

[6] Themistoklis Prodromakis, Boon Pin Peh, Christos Papavassiliou, and Christofer Toumazou. A versatile memristor model with nonlinear dopant kinetics. *Electron Devices, IEEE Transactions on*, 58(9):3099–3105, 2011.

[7] Ádám Rák and György Cserey. Macromodeling of the memristor in spice. *Computer-aided design of integrated circuits and systems, IEEE Transactions on*, 29(4):632–636, 2010.

[8] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[9] Frank Y. Wang. Memristor for introductory physics. *arXiv preprint*, arXiv:0808.0286, 2008.