MADX: Memristors-As-Drivers for Crossbar logic

Thomas Charles Sweeney

EE 679H
Engineering Honors
The University of Texas at Austin

December 12, 2017

Lizy Kurian John, Ph.D.
Department of Electrical and Computer Engineering
Supervising Professor

Jonathan Valvano, Ph.D.
Department of Electrical and Computer Engineering
Second Reader

Abstract

MADX: Memristors-As-Drivers for Crossbar logic

Thomas Charles Sweeney, Bachelor of Science
The University of Texas at Austin, 2017

Supervisor: Lizy Kurian John

Memristors have the potential to not only replace conventional memory, but also to open up new design possibilities because they store 1s and 0s as resistances rather than voltages. A memristor architecture that has attracted interest for its versatility and ease of integration with existing CMOS technologies is the crossbar array. In this paper, I modify the MAD scheme to create the MADX scheme for performing basic logic operations within a crossbar array. Then, I compare this scheme against two of the most well-known schemes, MAGIC and IMPLY. In the case study of a full-adder, both a one-bit and an 8-bit version, the MADX scheme achieves lower latency and substantially lower area requirements than both MAGIC and IMPLY. This is because it is more flexible about storing output values than either, does not destroy input values unlike IMPLY, and has more basic operations. In particular, it has XOR, which neither IMPLY nor MAGIC have and is useful for addition.

Table of Contents

Introduction

Memristors carry tremendous potential for reshaping memory, logic operations, and matrix multiplication, but work remains to be done to actualize that potential. Leon Chua first predicted a resistor with memory, the memristor, in his seminal 1972 paper [1]. In 2008, 36 years after Chua's original paper, there was a breakthrough by a team at HP that found memristive properties in a thin-film (5 nm) crosspoint Pt-TiO2-Pt cell [2].

The key properties of memristors that make them so attractive are that: (1) they are non-volatile [3], fast [4], and dense [5]; (2) they are compatible with the CMOS manufacturing process [3], [5]; and, (3), storing logical values in resistance rather than voltage creates new design opportunities to exploit because each cell can be used as both operand and operator. Several studies [5], [6], [7] have good discussions of these properties. The properties in (1) are the most important. Non-volatility means memristors can replace NAND flash as permanent storage devices with higher densities and much higher endurance (10^10 vs 10^5) [8]. High-density [5], combined with read and write times at least as good as DRAM [5, 4], raises the possibility of replacing DRAM and creates the opportunity for memory-intensive or non-Von Neumann architectures [9]. (2), compatibility with CMOS, means any limitations in memristor functionality can be buttressed by more complex CMOS elements. Finally, (3), the resistance property, has led to Boolean logic applications [6], [9]-[11], potential one-cycle dot-product multiplication [12], a variety of proposed machine learning accelerators [13]-[15], and even neuromorphic computing [16], [17].

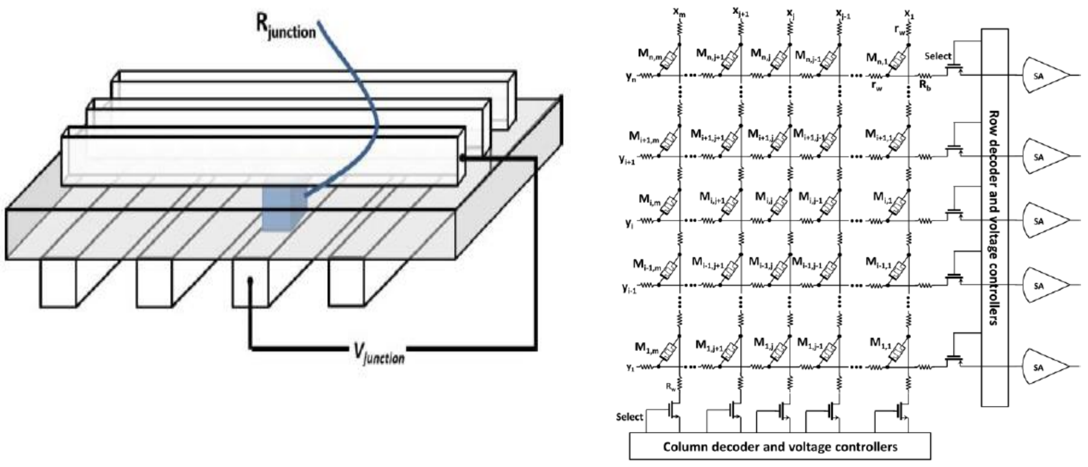
R_junction
V_junction
Select
Row decoder and voltage controllers
SA
Column decoder and voltage controllers

Figure 1: From [9]. (a) shows the physical crossbar structure while (b) shows the schematic representation of a crossbar. In (a) there is a top and a bottom layer of vias where each junction is connected by a memristor.

There have been a number of papers developing Boolean logic using memristors both inside and outside the crossbar structure. The original Memristors-as-Drivers (MAD) gate work [18], [19], IMPLY [10], [9], and MAGIC [11], [6] all have gate designs that are designed to also work outside the bounds of the crossbar in adders, multipliers, and dividers. Alternatively, papers like Xie et al. [20] have focused specifically on using the crossbar structure to perform any Boolean equation, but not in the context of logic in memory.

This thesis uses a popular memristor memory structure, the crossbar array, to perform logic operations in memory. The crossbar array, as shown in Figure 1, has drawn a significant amount of interest for its applications in memory, matrix multiplication, and logic. A number of schemes have been proposed for performing logic operations in memory of which IMPLY [10] and MAGIC [11] have gotten the most attention. These schemes have so far been limited; either only offering a few native operations, as in IMPLY and MAGIC, or requiring extensive hardware changes as with CRS [21], Zhang et al. [22], or the original MAD in crossbar schemes [18].

This thesis's contribution is MADX (Memristors-as-Drivers for Crossbar logic), a modification of the MAD gate crossbar framework [18]. MADX offers a greater range of basic operations than IMPLY or MAGIC, but avoids the additional hardware requirements and error vulnerability of the original MAD crossbar framework. These advantages are illustrated in a case study of an 8-bit full-adder implementation that has a lower latency and a smaller area than either the IMPLY or MAD frameworks. The MADX scheme is also much more flexible in terms of the area it needs as it can write the result of each operation into any cell in the crossbar.

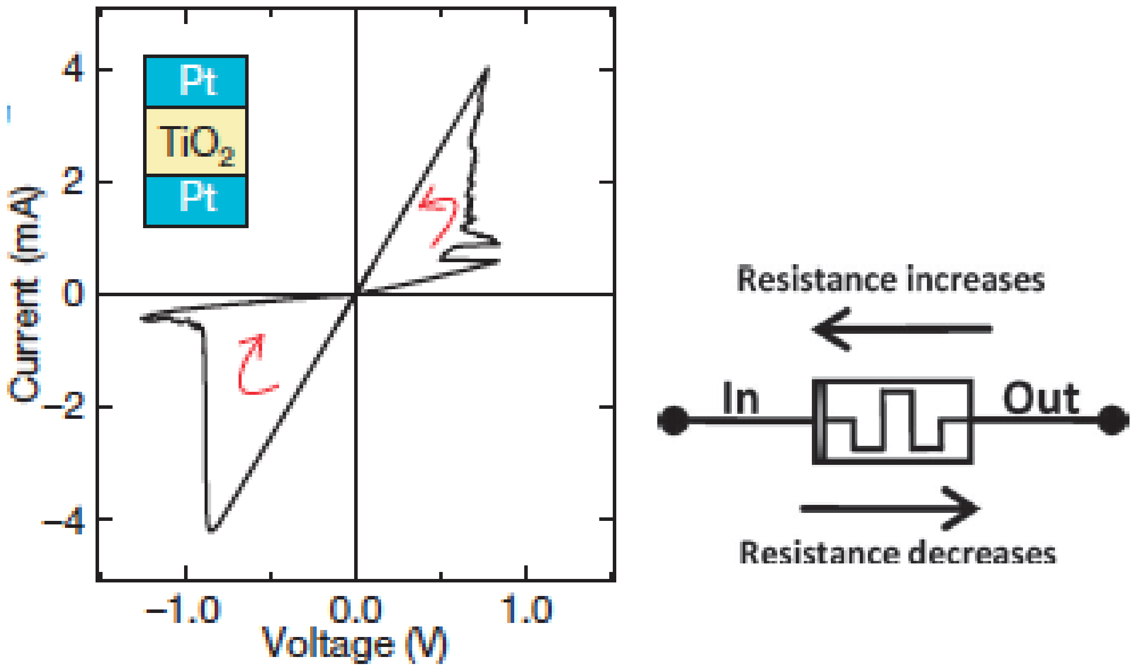

Figure 2a and 2b from [2] and [11]. (a) shows the distinctive pinched hysteresis loop or Lissajous figure that characterized the memristor IV relationship. (b) shows the circuit symbol and illustrates how the flow of current across the device induces switching depending on the memristor's polarity.

Introduction to Memristors

Memristor Basics

A memristor, in the simplest definition, is a two-terminal (one-port) device whose resistance is dependent on the state variable *w*. Its most distinctive feature is its IV relationship, the pinched hysteresis loop that crosses the origin as shown in Figure 2a. The circuit symbol is shown in Figure 2b. The simplest mathematical description, as outlined in [2] and [1] is shown in equations 1 and 2.

$$v = \mathcal{R}(w)i \tag{1}$$

$$\frac{\mathrm{d}w}{\mathrm{d}t} = i \tag{2}$$

w here is the state variable, a mathematical convenience for representing some internal physical change that occurs when current flows through the device. However, as Strukov [2] also notes, this simplest memristor definition has yet to be realized. Instead, what we are interchangeably calling memristors and memristive systems are usually described by the dynamic equations from Chua and Kang in 1976 [23]

$$v = \mathcal{R}(w,i)i \tag{3}$$

$$\frac{\mathrm{d}w}{\mathrm{d}t} = f(w,i) \tag{4}$$

where *w* is a state variable or a set of state variables. How one defines the functions R and f is crucial to the properties of the model, but a detailed discussion of the various models used is beyond the scope of this paper. Most papers use this model of a current-controlled memristor, but for Boolean logic, as in this paper, it is useful to use a voltage-controlled memristor model. For a voltage-controlled, time-invariant memristive device [24][1], the equations become

$$\frac{dw}{dt} = f(w,v) \tag{5a}$$

$$i(t) = G(w,v) \cdot v(t) \tag{5b}$$

With f(w,v) further described in the VTEAM [24] model as

$$\frac{dw(t)}{dt} = \begin{cases} k_{\mathrm{off}} \cdot \left(\frac{v(t)}{v_{\mathrm{off}}} - 1 \right)^{\alpha_{\mathrm{off}}} \cdot f_{\mathrm{off}}(w), & 0 < v_{\mathrm{off}} < v \quad (6a) \\ 0, & v_{\mathrm{on}} < v < v_{\mathrm{off}} \quad (6b) \\ k_{\mathrm{on}} \cdot \left(\frac{v(t)}{v_{\mathrm{on}}} - 1 \right)^{\alpha_{\mathrm{on}}} \cdot f_{\mathrm{on}}(w), & v < v_{\mathrm{on}} < 0 \quad (6c) \end{cases}$$

where k_{on}, k_{off}, α_{on}, and α_{off} are chosen constants and v_{on} and v_{off} are the threshold voltages. f_{off} and f_{on} are window functions that ensure the state variable w is between w_{on} and w_{off}. In this model, the state variable w will only change when the voltage is above the set voltage or below the reset voltage. In addition, in both this equation and the earlier current-controlled model, w is bounded such that it has a min and a max. The max and the min values for w correspond to two distinct resistances. These resistances can be assigned to logical '1' and logical '0' and are called Ron and Roff, respectively. Ron is the low resistance and Roff is the high resistance.

There has been some debate about whether devices we currently call memristors should be classified as memristors or memristive devices [24], but, similarly to other papers written on memristors for logic [6], [9]-[11], [20], this paper uses the two terms interchangeably.

Memristors For Boolean and Crossbar Logic

Since we are using memristors for Boolean logic, in a similar fashion to [6], [9], [20], we can first simplify the memristor model by making a couple of assumptions. First, we assume the memristors used only have the two states represented by the two different sloped lines in the hysteresis curve in Figure 1a. These two states, Ron and Roff, are mapped to the Boolean values of '1' and '0'. Two, we use a voltage-threshold memristor model, as outlined in [24], as voltage-threshold memristors are generally more useful for crossbars and digital logic in general. For the crossbar, voltage-threshold memristors are more useful because they will not switch states due to sneak path current. They are also more useful from a power and voltage supply perspective as can be seen from looking at the equation for resetting a current-threshold memristor

$$I(t) = \left|\frac{V_{rst}}{R(t)}\right| \geq |I_{th}| \tag{7}$$

When we write a logical 0 to a memristor (a high resistance), R(t) will increase during the writing process such that Vrst will have to be high enough that the current is still above the threshold cutoff for even a high resistance. However, high voltages increase the power consumption and increase the problem of sneak path resistances. As a result, there is a need for a voltage threshold cutoff, a physical property which has been shown, as noted in the VTEAM paper [24].

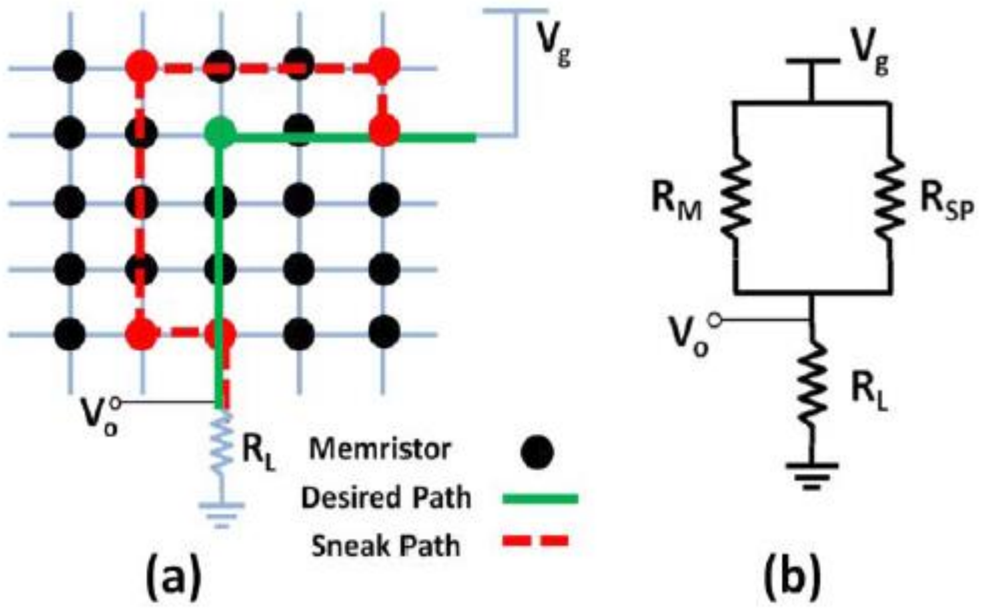

Figure 3: From [9]. (a) shows how current can leak. (b) shows a model of this behavior with Rm as the memristors resistance and Rsp as the sneak path resistance. (a) only shows one sneak path, but there are several additional sneak paths that will lower Rsp as they are in parallel.

Sneak path problems, as shown in Figure 3, are a key issue holding back crossbar arrays [25]. Sneak paths are alternative paths that current could travel through the crossbar that lie in parallel to the intended path. An equivalent circuit is shown in Figure 3b. Sneak paths greatly complicate the read operation in the crossbar because they make it difficult to tell the difference between high and low resistances. A wide range of solutions have been proposed, as shown in

[25] and [5]. Since the MADX scheme uses voltage sensing and a 1M crossbar structure, it is vulnerable to sneak paths, but not necessarily more vulnerable than MAGIC or IMPLY. In this thesis, we will generally assume the techniques outlined in [25] are used to mitigate the problem. We explicitly consider sneak paths in some places, but more work remains to be done to ensure this scheme is not vulnerable to them.

MAD Gates

Original MAD Gate

Memristors-As-Drivers (MAD) gates [18], [19], integrate CMOS with memristors in order to solve problems with fanout, signal degradation, and buffering issues as well as shorten the number of steps per logic operation as presented in Figure 4.

There are two different sense positions, V1 and V2, possible for the MAD gate, as can be seen in Figure 4. V1 is the sense position on the p-terminal of the second memristor that comes after the two memristors. V2 is the sense position on the n-terminal of the first memristor that comes before the two memristors.

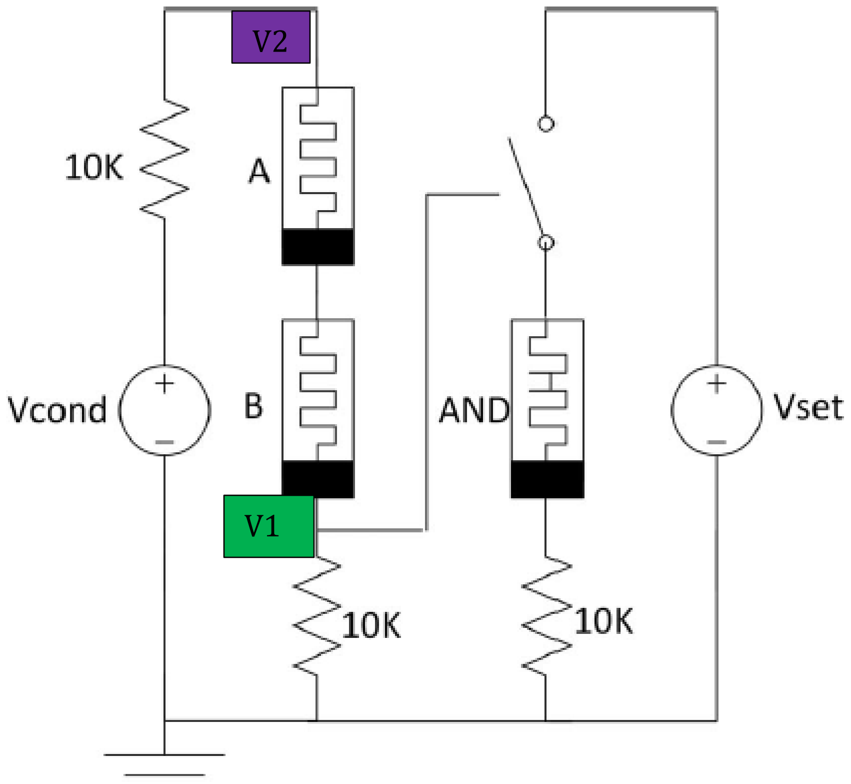

Figure 4: From [18]. MAD implementation of an AND gate. V1 and V2 values are shown in Table 1.

Table 1 shows the values for the two sense positions for Vcond = 1.5, Rg = 10kOhms, Ron = 1kOhm and Roff = 100kOhms. The setup in Figure 4 with V1 gating the output memristor is used for the AND, OR, and COPY operations is described by equations 8 and 9:

$$V1 = Vcond * \frac{(1-(Rs+Ra+Rb))}{(2Rs+Ra+Rb)} \tag{8}$$

$$Out = \begin{cases} 1 & V1 \geq Vthreshold \\ 0 & Otherwise \end{cases} \tag{9}$$

Where Rg is the resistance of the sense resistors used for voltage division labeled as 10k in Figure 4. *Vthreshold* is the voltage threshold of the switch.

AB	**V1 (V)**	**V2 (V)**
11	0.681818182	0.818181818
10	0.123966942	1.376033058
01	0.123966942	1.376033058
00	0.068181818	1.431818182

Table 1: Sense voltages for the various sense positions and input memristor combinations in a conventional MAD gate.

For an AND gate, the sensed voltage should only be greater than Vthreshold when both Ra and Rb are equal to Ron (logical 1), which is the low resistance state. As a result, we should choose a Vthreshold between 0.68 V (‘11’) and 0.12 V (’10’). For an OR gate, the same setup can be used, but the threshold voltage must change to be between 0.12 V (‘01’) and 0.07 V (‘00’). The COPY operation will only have 1 memristor instead of 2, so it is the same as equation (8) with Rb set to 0 Ohms. Both the OR gate and the COPY gate are shown in Figure 5.

For the NAND, NOR, and NOT operations, the same setup as in Figure 5 is used, but with the sense position set to V2, the purple label. This is demonstrated for the NOT gate in Figure 5. For a NAND gate, we can set the voltage threshold to be between 0.81 V (‘11’) and 1.38 V (‘10’). Similarly to the COPY operation, the NOT operation is just a special case of the setup with a single memristor instead of two, i.e. with Rb = 0.

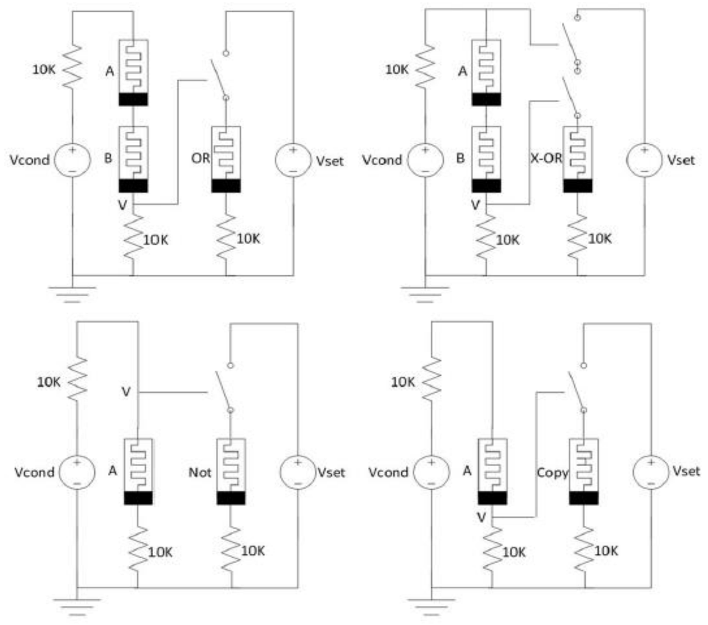

Figure 5: MAD OR, XOR, COPY, NOT gates from [18]

For the XOR and XNOR operations, we need to sense at both V1 and V2, as shown in Figure 5. As a result, there must be two switches instead of one, with both switches' threshold voltages chosen in accordance with Table 1. For instance, for the XOR gate, the switch connected to V1 should have a voltage threshold between 0.07 V ('00') and 0.12 V ('01'), and the switch connected to V2 should have a voltage threshold between 0.82 V ('11') and 1.38 V ('10').

A key contribution of this paper is the insight that we can also reuse some thresholds; for instance, the threshold for an AND gate can be reused as the threshold for a COPY gate. Similarly, the threshold for a NAND gate can be reused as the threshold for a NOT gate. Most importantly, the XOR gate can reuse the threshold voltage of the OR gate for V1 and the threshold voltage of the NAND gate for V2. Although this is not particularly relevant to the case of a static MAD gate, this flexibility becomes very important when we discuss the conversion to the crossbar array later on.

Conversion for Crossbar Logic

As shown in Figure 5a, the MAD gate needs to be modified to be more compatible with the crossbar structure so that the two memristors are now in parallel rather than in series. It is possible to use memristors in series in a crossbar array; however, that process is difficult and vulnerable to the sneak path problem. For instance, if we set the column containing Va in Figure 5b to Vcond and set the column containing Vb to ground then p and q would be connected in series. Guckert [18] notes that this can cause the memristors to switch. Recalling the memristor schematic from Figure 2b, if p was set to '1' (Ron) and q was '0' (Roff), then almost the entire voltage drop across the pair of memristors would be across q. Since this would be flowing into the p-terminal (the marked end), it could potentially cause q to switch to 1 from 0. This is true for current-threshold memristors, but, if Vcond is sufficiently small and we are using voltage-threshold memristors, it should not cause this behavior. However, using a smaller Vcond reduces the voltage differences between different combinations of p and q. In this series model, there would also be sneak-paths all the way up between the two columns. This means that there would be a significant amount of resistance in parallel. This problem could potentially be met by one of the standard sneak path solutions [25] of tying all of the unselected row lines to ground, but that is beyond the scope of this paper.

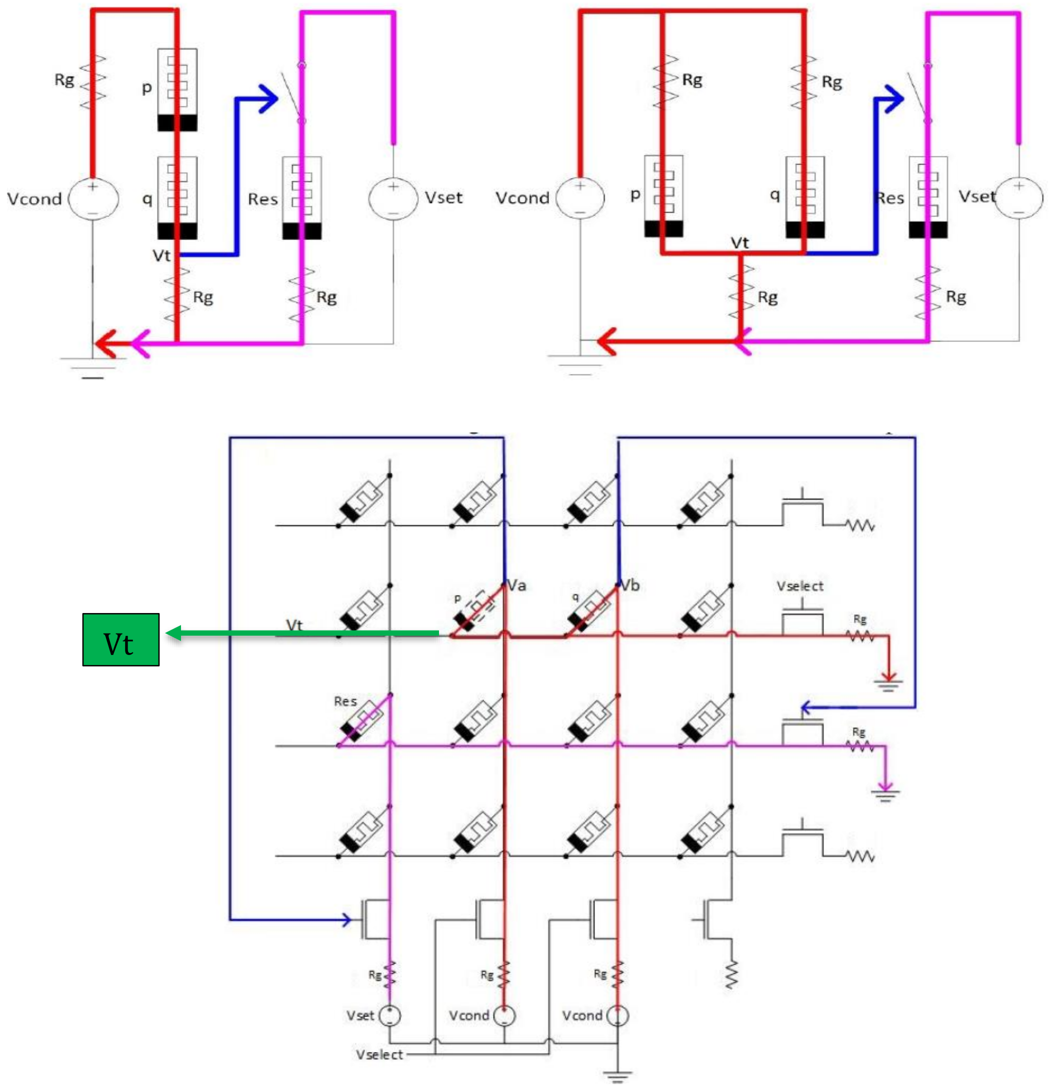

Figure 5: From [18]. (a) Conversion of the MAD gate into a crossbar compatible structure. (b) XNOR implementation of MAD gate in crossbar.

Guckert’s original way for performing the MAD operation for the XNOR gate is shown in Figure 5b. In the first step, the output memristor must be set to ‘0’. In the second step, Vcond is applied to the two columns containing the input memristors, and the two rows containing the input memristors are connected to ground. We will call this the sense operation. The values Va and Vb are latched in the controller. In the final step, the set operation, Va and Vb are used to gate a transistor connecting Vset to the column line containing the output resistor and to gate a transistor connecting ground to the row line containing the output resistor, respectively. If the values are greater than the threshold voltages for the transistors, the voltage will be connected across the output resistor and will set the value to 1. For other operations like AND and OR, we instead sense and latch Vt, rather than Va and Vb, during the sense operation and then use that value to gate one of the transistors during the set operation.

Limitations of MAD for Crossbar Logic

The original MAD scheme has problems in the form of additional hardware requirements and threshold sensitivity. For hardware, one issue is that the MAD gate scheme needs at least one transistor, two for XOR and XNOR, with a different threshold for every type of logical operation it wants to accomplish. Therefore, the total number of additional transistors is equal to the number of logic operations + 2. Another issue that arises from adding these extra transistors is the amount of MUXing required to tie all those transistors together. The issue with threshold sensitivity is that the thresholds for Va and Vb are very close together for some inputs of p and q, as shown in Table 2. For example, the Va for pq = 00 is 1.57 V and 1.58 V for pq = 01. Taking into account the parasitic resistance from the crossbar wires, sneak path problems, and stochastic error in both transistors and memristors, it is unrealistic to expect those two cases to be distinguishable.

The biggest issue with the MAD scheme is the required precision of the voltages, memristors, and transistors. If we take the numbers from Guckert's thesis [18], for an XOR operation with Vset of 2 V, Vcond of 1 V, Ron of 1KOhm, and Roff of 100KOhm, the transistor needs to be able to distinguish between 0.975 V and 0.982 V. Guckert suggests choosing Rg, Ron, Roff, and Vcond appropriately to maximize the difference in Vthreshold, but this is not a viable solution for several reasons. One, increasing Rg and Vcond increases the power consumption of the scheme. Two, increasing Vcond brings Vcond closer to Vset, increasing the probability of accidentally switching memristors. Three, increasing Rg and Vcond doesn't increase the gap between the threshold voltages by very much. Four, the sneak path error and memristor stochastic error are large enough that any design using 1M crossbar arrays should attempt to maximize the error tolerance of the scheme.

MADX

Key Modifications to MAD

Due to these problems, the key changes made in this thesis to the original MAD gate in a crossbar are: to use the same two transistors (the one on the row line and the one on the column line) for every gate and to only use Vt, instead of Va and Vb. To enable support for a greater range of operations, the MADX scheme also changes the row switches (the row-select transistors) into p-type transistors such that they only are closed for sensed voltages less than the threshold. This allows use of the less-than operation in addition to the greater-than operation, enabling the XOR and NOR operations. Using the same two switches limits the number of possible operations, but reduces the hardware changes. Solely using Vt increases the error tolerance of the scheme because the differences between the values of Vt for various inputs are greater than Va and Vb, as shown in Table 2. Using only Vt also enables column grounding [25] to reduce sneak path error and creates the possibility for parallelization.

p,q	Va(V)	Vb(V)	Vt(V)
00	1.57	1.57	0.06
01	1.58	0.97	0.65
10	0.97	1.58	0.65
11	1.14	1.14	0.91

Table 2: Voltages for the various possible sense positions in a crossbar array. Assuming ideal situation with no sneak path resistance for values of Rg = 2kOhms, Ron = 1k, Roff =100k, Vcond = 1.6V

Using Vt as the sole voltage in conjunction with changing the row transistors to p-types from n-types is more fault-tolerant, simplifies the logic, and still supports 4 basic operations. Looking at Table 2, the voltages for the sense positions can be examined and used to select two thresholds corresponding to the two switches. Using the OR gate as an example, we should select the threshold for the n-types to be .35 (midway between the Vt for '00' and '01'). If we select the threshold 0.78 V (midway between the Vt for '10' and '11') for the p-types, such that any Vt less than 0.78 V will cause writeback, then we now have a NAND gate. If we use Vt as the input for both the p-types and the n-types at the same time, then any Vt value between 0.35 and 0.78 will set the output value. Therefore, for this selection of threshold values, we can immediately realize 3 types of native operations: OR, NAND, and XOR. An important point is that for the OR and NAND cases we will only use one of the two switches and we will connect Vlow or Vhigh to the other switch, respectively. That should not be too costly because the memristor crossbar must already have that functionality in order to select any particular memristor. We can also realize at least one additional gate by looking at Table 3, the sensed voltages for a single memristor. Since the Vt for p = 1 of 0.64 is greater than threshold voltage of 0.35 V for the OR gate, we can also realize the COPY operation with the same threshold voltages and, hence, with the same transistors. The NOT operation can be realized with slightly more difficulty by NANDing p with a memristor pre-set to 1.

P	**Va (V)**	**Vt (V)**
0	**1.57**	**0.03**
1	**0.96**	**0.64**

Table 3: Sensed voltages for a single memristor. Ideal case with no sneak paths for values of Rg = 2kOhms, Ron = 1k, Roff =100k, Vcond = 1.6V. Vt for COPY operation highlighted in red.

pq	Vt(V) 1 SP	Vt(V) 2 SP
00	0.48	0.72
10	0.68	0.85
11	0.93	0.95

Table 4: Sensed voltages for cases with 1 worst-case sneak path and 2 worst-case sneak paths. SP stands for sneak path. Values of Rg = 2kOhms, Ron = 1k, Roff =100k, Vcond = 1.6V

Sneak Path Problem

Sneak paths pose a significant challenge not just to this scheme but to all memristor memory, in particular 1M crossbar arrays as used in this paper. As shown in Table 4, if there is just one worst-case sneak path (in a the worst-case sneak path, all the memristors in the sneak path are in the Ron, low-resistance state) the values change substantially from the ideal case presented earlier in Table 2, in particular for the '00' case. With two worst-case sneak paths, the Vt for the '00' case will be greater than '01' for the ideal case in Table 2, meaning the n-type transistors are no longer able to distinguish between '00' and '01'. Furthermore, there are always more than two sneak paths, and they all lie in parallel, reducing the worst-case resistance between the row lines, especially as the size of the array scales as in [25]. However, this is not a reason to reject the MADX scheme, as other Boolean logic-in-crossbar schemes like IMPLY [9] and MAGIC [6] are just as vulnerable to these issues. Furthermore, there are effective strategies for mitigating the problem, as discussed in [6], [25]. For instance, we can eliminate the sense resistors on the columns and ground the unselected columns, which showed good effects in [25]. Alternative crossbar structures like 1T1M (1 transistor 1 memristor) or 1D1M(1 diode 1 memristor) solve the sneak path problem as well. This thesis doesn't consider alternative memristive structures like 1T1M or 1D1M because, although they are not as vulnerable to the sneak path problem, they have significant drawbacks due to their added elements, and both IMPLY [9] and MAGIC [6] use 1M crossbars.

p,q,r	Va(V)	Vb(V)	Vc(V)	Vt(V)
000	1.57	1.57	1.57	0.09
100	0.98	1.58	1.58	0.66
110	1.15	1.15	1.59	0.92
111	1.24	1.24	1.24	1.07

Table 5: Sensed voltages for a 3-input operation. Ideal case with no sneak paths for values of Rg = 2kOhms, Ron = 1k, Roff =100k, Vcond = 1.6V.

N-input gates

The MADX design can realize 3-input OR gates, but cannot realize more than 2-input NAND gates. This can be demonstrated by comparing Table 2 and Table 5. In Table 5, the 3-input OR gate threshold voltage must be between 0.09 V and 0.66 V. Clearly, we can select a voltage that falls within that range and the range of 0.06 V and 0.61 V for the 2-input OR gate as demonstrated in Table 2. In Table 2, the NAND threshold voltage must be less than 0.91 V. However, in Table 5, the NAND threshold voltage for a 3-input NAND gate must be between 0.92 V and 1.06 V, eliminating the possibility of having a switch that satisfies both conditions. 4 or 5-input OR gates should be possible, but examining that question and how the 3+ input OR gate interacts with sneak paths is beyond the scope of this thesis. Standard sneak path mitigation techniques, as presented in [25], should help solve the problem.

Parallel Operations

Under the MADX scheme, it also becomes possible to perform multiple sense or multiple set operations in parallel, but not sense operations at the same time as set operations. The final set operation cannot be conducted in parallel with the sense operation because the Vset voltage will be high over all the memristors in the column, including whichever memristor is connected to the row we are trying to sense. This will change Vt and might even change the value of the memristor that connects the column to the sensed row. Furthermore, we cannot perform the sense operation in parallel on different columns and rows because that would affect the values for the row we are sensing. However, there still exists the possibility of setting multiple elements in a column or performing multiple sensing operations for elements in different rows, but in the same columns, in a single cycle. As long as the sensing generates values that are compatible with the normal operation case, then the set operation should be able to write back as many results in the same column as we can generate.

However, sneak paths could derail this just as they could derail the normal operation case, and the controller logic for handling this parallelism would need to be more sophisticated. Let’s say we are doing two operations, with sense operation being conducted with inputs of p,q = 0,0 and the other with p,q = 1,1. In the ideal case, one row line (00) would be equal to 0.06 V and the other row line would be equal to 0.91 V as shown in Table 2; however, there could be as little as 2kOhms (2 * Ron) of resistance in between the two row lines due to sneak paths. That would cause the low voltage to be pulled up to 0.39 V and the high voltage to be pulled down to 0.751 V. Examining Table 2, those values are still within values that could be captured – the threshold for the OR gate would just have to be between 0.39 V and 0.65V and the threshold for the NAND gate would need to be between 0.65 and 0.75 V. However, those are very slim margins for error. It seems possible to perform operations in parallel for sets of inputs on the same columns, but it needs further exploration.

Parallel operations might even help mitigate the sneak path problem because the selected row lines will all be less than 0.91 V. The voltage of the unselected row line lying on the sneak path must always be greater than the voltage of the selected row line since current is flowing through at least two more memristors between the unselected row line and the selected row line.

The selected lines will always have a lower voltage than the voltage of any floating, unselected row line lying along a sneak path. As a result, adding parallel operations should generally help solve the sneak path problem.

MADX Flexibility

Since the MADX scheme does not require the addition of additional circuitry to the crossbar array beyond the resistors added in IMPLY, it could be integrated with IMPLY. The best way to combine the two is beyond the scope of this paper, but it illustrates the flexibility and simplicity of the MADX scheme.

The MADX scheme and the MAD scheme it derives from can be described as voltage division and sensing schemes. The name underscores the simplicity and flexibility of the concept. By dividing the sense and set steps, MADX enables much more flexibility in terms of what area it needs to use and where the inputs are in memory. This flexibility might be the most important aspect of the whole design and is demonstrated in the Comparison to Alternatives section through the case study of an adder.

Limitations

There are some important caveats to keep in mind both for this thesis and for the other schemes that do logic in memory. Using memristors for logical operations greatly increases the amount of writing to them, as the authors of [6] pointed out. They note that memristor endurance is still low, 10^10 [8], so this is a substantial drawback. However, they also note that memristor endurance should increase substantially in the coming years, citing [26]. Moreover, sneak paths and the high stochastic of memristors remain challenges. As shown in [25], the size of the difference in voltage drops across an on and an off memristor is reduced to a fifth of its value in the ideal case by the time we reach an array size of 256 in an array with some 1s, declining to practically 0 by the time we get to an array size of 1024. Parasitic resistance on the row and column lines can also cause problems in getting accurate values as the size of the array scales. The MADX scheme itself is limited in that it can only perform logical operations between memristors that are on the same row. It also requires the addition of resistors to the crossbar array as in IMPLY. Moreover, questions about how to best implement the controller logic remain.

Comparison To Alternatives

Full Adder Case Study

A 1-bit half-adder is described by the equations

$$\text{SUM} = \text{A XOR B} \tag{10a}$$

$$\text{CARRY} = \text{A AND B} \tag{10b}$$

The MADX design does not implement AND as a basic operation, but it can be done in two steps by doing

$$\text{CARRY} = \text{NOT(A NAND B)} \tag{11}$$

As a result, even without parallelization, the total number of operations for a half-adder in the MADX scheme is 3 operations, which, at three cycles per operation, takes 9 cycles. For the 1-bit full-adder, there are three inputs Cin, A, and B and two outputs, whose relationship is described by the equations

$$\text{SUM} = \text{Cin XOR (A XOR B)} \tag{12a}$$

$$\text{CARRY} = \text{A AND B OR (Cin AND (A XOR B))} \tag{12b}$$

The SUM function takes 2 steps while the CARRY function takes only 5 steps because we can reuse the A XOR B output generated during the SUM function. A detailed walkthrough is as follows. First, the output of A XOR B is generated and stored on an output row line somewhere that has Cin on it as well. Then, that output is XORed with Cin to generate SUM for 2 total steps. For the CARRY operation, the output of A XOR B generated in the first step is ANDed with Cin by performing NOT(NAND) as with the half-adder. A similar procedure is followed for A AND B. Then those two outputs are ORed together to give the CARRY output for 5 total steps. In total, there are 7 serial operations for each 1-bit full adder. Since MADX gates can writeback the output to any part of the crossbar structure, the scheme avoids the substantial initializing required by IMPLY and MAGIC.

Generalizing this result to an 8-bit adder, MADX requires 7n operations for an n bit adder, which equals 56 operations for an 8-bit full-adder. In terms of area, it needs at least one free memristor on the same line as Cin for every Cin. It also needs two free memristors on an arbitrary row line in which to place the result of A AND B and CIN AND (A XOR B). For the AND operation, there also should be a row line somewhere with a memristor preset to 1 and a free memristor. This gives a total of 1 + 2 + 2 for 5 total functional memristors. We need only preset the '1' used for doing the NOT operation, which adds one additional cycle. Even without parallelization, which this thesis noted early should be possible, MADX gates are substantially better than MAGIC and IMPLY gates, as shown in Table 6. These improvements are a direct result of the increased number of basic operations provided and the flexibility of writing the result to wherever we want it. Moreover, IMPLY and MAGIC gates require all the inputs to be on the same line, while MADX only requires each operand in any particular operation to be on the same line.

Scheme	Latency	Area (# functional memristors)
IMPLY Base [10]	**89N**	**4**
IMPLY Serial [9]	**20N**	**2**
IMPLY Parallel [9]	**5N + 18**	**6N - 1**
MAGIC Conv., area optimized [6]	**15N**	**5**
MAGIC Conv., latency optimized[6]	**12N + 1**	**11N - 1**
MADX Serial	**7N + 1**	**5**

Table 6: Latency and area comparisons for an N-bit adder

MADX is better than a variety of MAGIC and IMPLY schemes in terms of latency and dramatically better in terms of area, as shown in Table 6. IMPLY Parallel [9] does achieve better latency in terms of number of operations, but it requires switches between the different rows of the crossbar. In defense of IMPLY and MAGIC, it should be noted that MADX operations take three cycles while IMPLY and MAGIC operations only take two cycles. Therefore, the MADX serial scheme is only slightly better than MAGIC in terms of latency; however, it still offers the best combination of latency and area while being very flexible.

Alternative Logic in Memory Schemes

A number of alternative logic designs have been proposed. The two types that have attracted the most attention are the memristor-aided logic (MAGIC) scheme proposed by Kvatinsky et al. [11], expanded on in Talati et al. [6], and the IMPLY scheme proposed in 2012 [10] and improved in 2014 [9]. These schemes are explained below and a comparison of them with MADX across some key categories is shown in Table 5. Both the IMPLY scheme and the MAGIC scheme, notably developed by the same group, are the only schemes that perform logical operations in a 1M crossbar without additional CMOS logic. This gives them substantial benefits with regards to size and ease of integration. Two other schemes that have achieved some recognition are CRS [21] and Zhang et al. [22]. However, they are not considered here because they require substantial alterations to the basic crossbar hardware. A future area of research would be to do a more thorough comparison with these alternative schemes.

Scheme	Voltages Required	Basic Ops	Steps per Op	Memristors per op	Destructive Operation	Input/Output restrictions
MAGIC [6] [11]	1 (Vo)	NOR	2	3	No	Inputs and Output same row
IMPLY [9], [10]	3 (Vcond, Vset, Vreset)	IMPLY (with FALSE)	2	2 (destroys an input)	Yes	Inputs and Output same row
MADX	3, same as IMPLY	5	3	3	No	Inputs same row, Output anywhere

Table 5: Comparison of MADX scheme to alternatives. MADX's 5 basic operations are NOT, COPY, NAND, OR, and XOR

The MAGIC approach solely uses memristors to perform logic operations. The MAGIC scheme can be used to perform all logic operations outside of a crossbar, but can only be used to perform NOR operations within a crossbar due to the constraints imposed by the crossbar. In the MAGIC scheme, two or more memristors on the same bit or word line can write to an output memristor on the same bit or word line. This is a key drawback when compared to the MAD structure where the result can be placed on any word or bit line. Innovations in the 2016 paper demonstrated a crossbar can have the ability to perform those NOR operations in parallel in a column-wise fashion, greatly increasing the efficiency of the scheme [6]. Even with this parallelization, the MAGIC gate is worse in terms of latency and substantially worse in terms of area when compared with MADX for the full-adder case study due to its lack of flexibility and limited basic operations.

The IMPLY scheme uses memristors without any additional CMOS logic to perform material implication, and it is the most established and popular gate design. Despite only performing material implication, when combined with FALSE (a logical function that only yields 0) it can perform any Boolean function [27]. It was the original inspiration for the MAD gates and, like MAD, adds resistors to the rows and columns.

Methodology

This paper assumed a 1M crossbar. Extensions of the MADX scheme to 1T1M or 1D1M crossbars are beyond the scope of this paper. The voltage values for the tables were generated using NGSPICE. The VTEAM Verilog-A model [24] was used to model the behavior of the memristors. We used the parameters used in MAGIC [11] for our VTEAM model. We modeled the memristors as linear. Non-linear models could be used in the future to help eliminate the sneak path problem. However, the thresholds of non-linearity would have to be chosen with care if they were to be compatible with this scheme; Vcond must be in the linear region, but also be less than the voltage used to set the memristor.

Prior Work

Memristor research can be classified into research done into understanding the physics and improving the basic properties of memristors and research done into the applications of memristors. The former includes improving the endurance and read and write times of memristors in addition to developing better mathematical models for memristor behavior. Regarding the latter, there are a wide range of potential applications, but this section will focus on applications having to do with crossbar arrays and Boolean logic, as well as some discussion of the more novel applications of memristors.

Memristor Physics

While this paper and many others have focused on the applications of memristors, there has been a tremendous amount of work put into understanding the physics and improving the material science behind memristors [5]. This has included the development of new types of memristors in addition to the original TiO_2 version. These papers have also lead to the development of more accurate models for memristive behavior as in VTEAM [24]. As many researchers on the applications of memristors are not actually using a physical memristive device, the development of memristor models has been and will continue to be crucial to memristor development. With regards to improving the basic properties of memristors, experiments have led to sub 1 ns read and write latencies [4] and durability of up to 10^10 write cycles [5].

Memristor Applications

The research into the applications of memristors has touched on a massive number of topics. Like the other fundamental circuit elements did, the memristor will likely have an impact across all elements of circuit theory and design. Even if we restrain our scope to just a digital, computing perspective, summarizing the work is a massive endeavor. However, three areas do seem to stand out.

Memristor crossbar structures, as shown at the beginning of this thesis in Figure 1a, offer tremendous potential for improving every layer in the memory hierarchy, speeding up dot-product matrix multiplication, and realizing processing-in-memory (PIM). The crossbar array is capable of performing dot-product vector multiplication in one step [12]. This property is tremendously valuable to modern machine learning algorithms as convolution and inference in neural nets require extensive matrix multiplication. Crossbars have also been successfully fabricated at a small scale and used for training simple classifiers so they are probably the most currently practical of the memristor research areas [28].

Resistive RAM (RRAM), memory using memristors, takes the form of a crossbar structure and has the potential for order of magnitude improvement over conventional memory in terms of density [25],[29],[30] in addition to being non-volatile. This and the one-cycle dot-product vector multiplication outlined above have together generated significant enthusiasm for building memories and accelerators designed specifically for machine learning. PipeLayer [13], PRIME [14], and ISAAC [15] showed in simulation how large-scale RRAM memories with PIM capabilities could achieve several orders of magnitude improvement over even the best conventional accelerators like DaDianNao [31]. The important caveat is that those papers were just simulation. However, there have been several papers that have realized more limited physical versions of some machine learning algorithms [16]. Another factor speeding adoption in the machine learning space is that memristors are better suited for storing weights in neural networks (NNs) than they are for storing conventional data because NNs are more error tolerant.

A final field is the application of memristors to neuromorphic circuits. As outlined in Hamdioui et al. [7], memristors have been used as weights in conventional artificial neural networks (ANNs) [16] and as pseudo-synapses in spiking neural network architectures that more closely mimic biological systems [17].

A final, fascinating area is the discovery of memristive properties in nature. Venus flytraps [32] and amoebas [33] have been shown to have some memristive properties. Memristors may be a useful framework for understanding hysteretic behavior in nature.

Proposed Future Work

There are a number of future research avenues for MADX. The first is investigating this design’s potential for parallelism and mitigating the sneak path problem. The effect of parasitic resistance on this design should be explored. The timing and power consumption of the scheme should also be investigated. The next design step is implementing the controller and routing logic to ensure the MADX scheme is compatible with conventional crossbar hardware and to get an accurate picture of the total costs of the design. Finally, manufacturing a physical prototype that uses this scheme is the ultimate goal. Additionally, a speculative route would be exploring possible integration with IMPLY logic.

Conclusion

In this paper, we extend and modify a design for Boolean logic, the MAD gate from Guckert’s 2016 thesis [18], to make it more compatible for integration into a crossbar array, creating the MADX scheme. This scheme leverages voltage thresholds and separates the sense and set operations. It supports a wider range of basic operations than IMPLY and MAGIC and is more flexible than either. Those advantages are reflected in the case studies of a 1-bit half adder and an 8-bit full adder where the MADX scheme has less latency, consumes significantly less area, and is much more flexible about where the area needs to be allocated.

Memristors and memristive devices can be manufactured more densely [5], consume less power, and could have quicker read and write times than current DRAM [4]. At the very least, they should be able to replace some NAND flash and hard-drive functionality. However, the key challenges regarding sneak paths, more frequent errors, integration of voltage sources, and manufacturing are difficult and will take a significant length of time to solve [7]. In the interim, IC designers and computer architects should continue to strive to explore how memristors can both augment conventional designs and open up entirely new avenues of research. By demonstrating the potential of memristors, we can hopefully also help spur additional research into the elements holding them back.

Bibliography

[1] L. O. Chua, "Memristor-The Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, pp. 507-519, 1971.

[2] 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.

[3] J. Borghetti *et al.*,“A hybrid nanomemristor/transistor logic circuit capable of self-programming,” *Proc. Nat. Acad. Sci.*, vol. 106, no. 6, pp. 1699–1703, 2009.

[4] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, “Sub-nanosecond switching of a tantalum oxide memristor,” *Nanotechnology*, vol. 22, pp. 485203-1–485203-7, Nov. 2011.

[5] J. J. Yang, D. B. Strukov, and D. R. Stewart, “Memristive devices for computing,” *Nature Nanotechnol.*, vol. 8, pp. 13–24, Jan. 2013.

[6] Talati et al., "Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC)," *IEEE Trans. Nanotechnology*, Vol. 15, No. 4, pp. 635-649, Jul. 2016.

[7] Hamdioui et al., "Memristor For Computing: Myth or Reality?," in *Design, Automation, and Test in Europe (DATE), 2017*.

[8] J. J. Yang *et al.*,“High switching endurance in TaOx memristive devices,” *Appl. Phys. Lett.*, vol. 97, no. 23, pp. 232102-1–232102-3, 2010.

[9] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 22, pp. 2054-2066, 2014.

[10] S. Kvatinsky, A. Kolodny, U.Weiser, and E. Friedman, “Memristor-based IMPLY logic design procedure,” in *Proc. IEEE 29th Int. Conf. Comput. Des.*, Oct. 2011, pp. 142–147.

[11] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC Memristor Aided LoGIC," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 61, pp. 1-5, 2014.

[12] M. Hu *et al.*, “Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Vector-Matrix Multiplication”, in *DAC*, 2016.

[13] L. Song, et al. “Pipelayer: A pipelined reram-based accelerator for deep learning,” in *HPCA*, 2017.

[14] P. Chi *et al.*, “Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.

[15] A. Sha_ee *et al.*, “Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.

[16] F. Alibart, E. Zamanidoost and D. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature communications,* vol. 4, 2013.

[17] A. Serb, J. Bill, A. Khiat, R. Berdan, R. Legenstein and T. Prodromakis, "Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses," *Nature Communications,* vol. 7, no. 12611, 2016.

[18] L. Guckert, “Memristor-Based Arithmetic Units,” Ph.D. dissertation, Dept. Elect. Eng., Univ. of Texas., Austin, TX, 2016.

[19] L. Guckert and E. E. Swartzlander, Jr., "MAD Gates - Memristor Logic Design Using Driver Circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 99, pp.1, Apr. 2016.

[20] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui and K. Bertels, "Fast Boolean Logic Mapped on Memristor Crossbar," *IEEE International Conference on Computer Design,* pp. 335-342, 2015.

[21] Y. Yang, J. Mathew, S. Pontarelli, M. Ottavi and D. K. Pradhan, "Complementary Resistive Switch-Based Arithmetic Logic Implementations Using Material Implication," *IEEE Transactions on Nanotechnology*, Vol. 15, pp. 94-108, Jan. 2016.

[22] Y. Zhang; Y. Shen; X. Wang; L. Cao, "A Novel Design for Memristor- Based Logic Switch and Crossbar Circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers,* Vol. 62, pp.1402-1411, May 2015.

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

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

[25] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, “Memristor-based memory: The sneak paths problem and solutions,” *Microelectron. J.*, vol. 44, pp. 176–183, Feb. 2013.

[26] J. Nickel, “Memristor material engineering: From flash memory replacement

towards a universal memory," in *Proc. IEEE IEDM Adv. Memory Technol. Workshop,* Oct. 2011, pp. 142–147.

[27] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, Apr. 2010.

[28] M. Prezioso et al. , "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, 2015, vol. 521, p. 61–64. DOI:10.1038/nature14441

[29] International technology roadmap for semiconductors. URL http://www.itrs2.net/

[30] M. Kryder, C. Kim, After hard drives-what comes next?, *IEEE Transactions on Magnetics,* 2009l 45 (10) (2009) 3406–3413.

[31] Y. Chen *et al.*, Dadiannao: A machine-learning supercomputer," in *Proc. MICRO,* 2014.

[32] A. G. Volkov, C. Tucket, J. Reedus, M. I. Volkova, V. S. Markin, and L. Chua, "Memristors in plants," Plant Signal. Behav., vol. 9, no. 2, pp. e28152(1)–e28152(8), Feb. 2014

[33] Y. V. Pershin, S. L. Fontaine, and M. D. Ventra, "Memristive model of amoeba's learning," Phys. Rev. E, vol. 80, no. 2, pp. 021926(1)–021926(6), Jul. 2010.