Design of Compact Memristive In-Memory Computing Systems using Model Counting

Dwaipayan Chakraborty Sumit Kumar Jha
Computer Science Department
University of Central Florida
4000 Central Florida Blvd
Orlando, FL 32816
Email: {dchakra, jha}@eecs.ucf.edu

***Abstract*—Crossbars of nanoscale memristors are being fabricated to serve as high-density non-volatile memory devices. The flow of current through memristor crossbars has been recently used to perform in-memory computations. However, existing approaches based on decision procedures only scale to the simplest circuits such as one-bit adders and other approaches employing decision diagrams produce large crossbar designs.**

In this paper, we present a new method for synthesizing compact combinational circuits using nanoscale crossbars. Our synthesis procedure exploits a symbolic representation of Boolean functions and employs model counting to guide a simulated annealing based search procedure. The proposed method creates crossbars that are up to about 6.3 times more compact than crossbars synthesized using decision diagrams. Our approach can scale to problems at least 4 times larger than the approach based on quantified decision procedures.

I. Introduction

Nanoscale non-volatile memristor crossbars provide an ideal framework for in-memory computing. Data can be readily stored on high-density non-volatile memristor crossbars and computation can then be performed on the same fabric using the switching of nanoscale memristors and the flow of current through the nanowires in the memristor crossbar [1]. Such in-memory computing frameworks provide a natural solution for data-intensive computing workloads.

Memristive circuits, whose topologies are not restricted to those of memristor crossbars, have been used to perform Boolean computations [2], [3]. More recently, memristive circuits that obey the topological constraints imposed by memristive crossbars have been designed for computing Boolean functions via two competing approaches: decision procedures for quantified Boolean formula [4] and Boolean decision diagrams [5]. The first approach based on quantified constraint solving creates compact crossbars but has not been reported to scale to circuits larger than a one-bit adder [6]. The second approach based on algorithmic layout of Boolean decision diagrams has been used to synthesize circuits as large as 128-bit adders but a large number of memristors (92% for a 4-bit adder) in the designed crossbar are turned off and not used during computations; hence, the designed crossbars are large in size. In this paper, we make the following contributions:

- We present a new synthesis approach that seeks to combine the scalability of the decision diagram method [5] with the compactness of the decision procedure based approach [4], [6]. Our algorithm uses model counting to determine the number of satisfiable instances for the Boolean formula representing the symmetric difference of the target Boolean function and the function computed by a crossbar to guide a simulated annealing algorithm over the search space of all possible crossbar designs.
- We demonstrate the success of our algorithm by synthesizing the *smallest* known crossbar circuits implementing a 4-bit adder and a 4-bit comparator. Our method produces adders that are up to *3 times faster* and *6.3 times smaller* than the state-of-the-art [3], [5].

TABLE I
Comparison of the size of our memristor crossbar for n-bit addition with existing crossbar computing approaches.

#bits / input	NNF-based [1]	SMT-based [4], [6]	BDD [5]	Our method	Prior best/ Our method
2	64×64	Time Out	8×5	4×4	$\mathbf{250}\%$
3	184×184	Time Out	15×9	6×4	$\mathbf{562.5}\%$
4	472×472	Time Out	21×12	8×5	$\mathbf{630}\%$

II. Related Work

Memristors are passive nonlinear two-terminal electrical devices whose resistances can be controlled by the passage of current through them and the resistances do not change when the current is removed [7], [8]. These devices have been exploited for performing arithmetic computations [9], [10], logical computation [11], and for neuromorphic computing [12]. Boolean Decision Diagrams have been used to compute Boolean functions using memristors [13]. This method maps a netlist to a circuit of 2-to-1 multiplexers implemented using memristors and inverters implemented using CMOS. Majority Inverter Graphs (MIGs) [14], [3] have been used to perform Boolean computations using two different elementary operations: the material implication (IMP) and the majority operation (MAJ).

Different physical manufacturing processes, including biological self-assembly, have been used to fabricate high-density crossbars of nanoscale memristors. The approaches discussed earlier in this section are not directly applicable to such

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

structurally constrained crossbars. Methods based on quantified decision procedures have been suggested to compute logical formula using sneak paths in memristor crossbars [4], [6]. Recently, algorithms have been suggested for laying out Boolean decision diagrams on nanoscale crossbars and using such crossbars for performing Boolean computations [5].

III. Our approach

Our approach relies on approximating the distance between the Boolean formula computed by a candidate crossbar design and the target Boolean formula using model counting [15]. We begin by building an abstract notion of a crossbar design for implementing Boolean computations.

Definition 1. Crossbar *A (memristor-based) crossbar is a 3-tuple* $\mathbb{C} = (\mathbb{M}, \mathbb{W}_r, \mathbb{W}_c)$ *where*

- $\mathbb{M} = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l1} & m_{l2} & \dots & m_{ln} \end{pmatrix}$ *is a two-dimensional array of memristors with* l *rows and* n *columns, where* $m_{ij} \in [0, 1]$ *denotes the state of the memristor connecting row* i *with column* j*;*
- $\mathbb{W}_r = \{r_1, \dots, r_l\}$ *is the set of horizontal nanowires such that wire* r_i *provides the same input voltage to every memristor in row* i*.*
- $\mathbb{W}_c = \{c_1, \dots, c_n\}$ *is the set of vertical nanowires such that wire* c_j *provides the same input voltage to every memristor in column* j*.*

The memristor $m_{ij} = 0$ is said to be in the high-resistance OFF state while $m_{ij} = 1$ denotes the low-resistance ON state.

Definition 2. Crossbar Design *A crossbar design* $\mathcal{D}(\mathbb{M})$ *maps each memristor* m_{ij} *in the crossbar* $\mathbb{M}$ *to one of the following: an input Boolean variable* $v_1, \dots v_n$*, its negations* $\neg v_1, \dots \neg v_n$ *or the logical constants* $True$ *or* $False$*.*

A visualization of a crossbar design with four horizontal nanowires (rows) and three vertical nanowires (columns) is shown in Figure 1. For a memristor crossbar with l rows, n columns and v different input variables, each memristor can be mapped to $2v + 2$ different values. Hence, the number of possible crossbar designs is as large as $(nl)^{2v+2}$. For example, our 4-bit adder using 8×5 crossbar has about 10^{28} designs. Hence, an enumerative analysis is infeasible and we pursue a combination of symbolic model counting and simulated annealing for searching the space of all possible designs.

Our approach is shown in Algorithm 1. In line 1, we first pick a random crossbar design of size l rows and n columns. Each memristor in the design is mapped to either $True$, $False$, one of the variables $v_1, \dots v_k$ or one of the negated variables $\neg v_1, \dots, \neg v_k$ using a uniform random distribution. For each crossbar design $\mathcal{D}$, we assume that a flow of current is injected into the lowermost horizontal nanowire (or row) and then we compute the values of the memristors in the crossbar design that cause the flow of current to reach the topmost horizontal nanowire (or row) of the memristor crossbar. In line 2 of the algorithm, we symbolically compute the Boolean formula representing the values of the memristors under which a flow reaches the topmost nanowire of the crossbar and denote it by BooleanFlow($\mathcal{D}_1$). Let $r_i^{(t)}$ denotes the flow value of row i at time t, and $c_j^{(t)}$ denotes the flow value of column j at time t. At $t = 0$, only the first row has flow i.e. $r_1^{(0)} = True$ and all other rows and columns are set to $False$. For all $t > 0$,

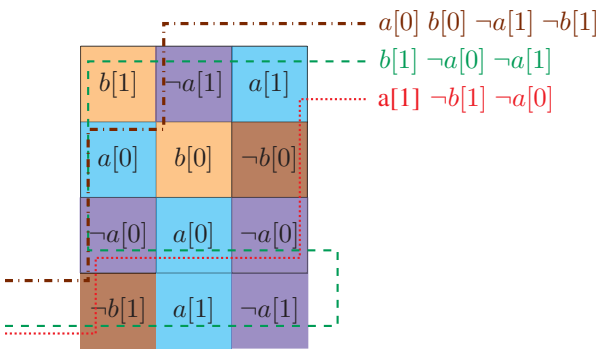

Fig. 1. A 4×3 crossbar design for computing the most significant bit of the sum of the inputs $a[0:1]$ and $b[0:1]$. Each memristor in the crossbar is labelled with the value that should be stored in the memristor – either a bit of the input a (colored light blue), a bit of the input b (colored light brown), negation of a bit of input a (colored deep blue) or negation of a bit of input b (colored deep brown). A flow of current is injected into the lowest horizontal nanowire and the logical values of the memristors control its flow to the topmost horizontal nanowire. The dotted red line indicates how the flow of current from the lowest nanowire to the topmost nanowire occurs along this path if the most significant bit (MSB) of the input a is 1 and the least significant bit (LSB) of both the inputs is 0. The dashed green line shows the flow when the MSB of the input b is 1 and the LSB of both the inputs is 0. The brown dashed-dotted line shows the flow of current when both the MSBs of the two inputs are 0 but both the LSBs of the two inputs are 1.

Algorithm 1: Crossbar Synthesis Algorithm

Input : Target Boolean Formula ϕ over variables $\{v_1, v_2 \dots v_k\}$
Size of crossbar $\mathbb{C}$: l rows and n columns
Initial temperature for simulated annealing T
Cooling rate c

Output : Crossbar Design $\mathcal{D}(\mathbb{M})$ mapping each memristor $m_{ij} \in \mathbb{M}$ to the set $\{True, False, v_1, \dots, v_k, \neg v_1, \dots, \neg v_k\}$

1 $\mathcal{D}_1 \leftarrow$ PickRandomCrossbarDesign($l, n, v_1, \dots v_k$)
2 $\mathcal{B}(\mathcal{D}_1) \leftarrow$ BooleanFlow($\mathcal{D}_1$)
3 $\Delta_1 \leftarrow$ ModelCount($\mathcal{B}(\mathcal{D}_1) \oplus \phi$)
4 **while** $\Delta_i > 0$ **do**
5 $\quad \mathcal{D}_{i+1} \leftarrow$ PerturbCrossbarDesign($\mathcal{D}_i, \phi$)
6 $\quad \mathcal{B}(\mathcal{D}_{i+1}) \leftarrow$ BooleanFlow($\mathcal{D}_{i+1}$)
7 $\quad \Delta_{i+1} \leftarrow$ ModelCount($\mathcal{B}(\mathcal{D}_{i+1}) \oplus \phi$)
8 $\quad$ **if** $rand(0, 1) < e^{-(\Delta_{i+1} - \Delta_i)/T}$ **then**
9 $\quad\quad i \leftarrow i + 1$
10 $\quad$ **end**
11 $\quad T \leftarrow c \times T$
12 **end**
13 **Return** crossbar design $\mathcal{D}_i$

we define the following transitions for each nanowire based on the ability of turned-on memristors to sort their horizontal and vertical nanowires.

$$\forall i \in \{2,\ldots,n\}, r_i^{(t+1)} \iff \left(r_i^{(t)} \vee \bigvee_{1\le j\le n} (m_{ij} \wedge c_j^{(t)})\right)$$

$$\forall j \in \{1,\ldots,m\}, c_j^{(t+1)} \iff \left(c_j^{(t)} \vee \bigvee_{1\le i\le l} (m_{ij} \wedge r_i^{(t)})\right)$$

The above transitions of the rows and columns in the crossbar can be represented using Boolean functions and described succinctly using Boolean Decision Diagrams (BDDs), And Inverter Graphs (AIGs) or other representations.

Line 3 of the algorithm computes the approximate number of satisfiable instances Δ_1 to the Boolean formula corresponding to the symmetric difference of the target Boolean formula ϕ and the formula corresponding to the computation performed by the crossbar design $\mathcal{D}_1$. Several competitive implementations of approximate model counting algorithms are available and our approach is agnostic to the choice of the model counting strategy as long as the algorithm produces a count of 0 feasible models only for unsatisfiable formula [15].

The loop in line 4 through line 12 continues to perturb the crossbar design, evaluate the function that this perturbed design computes and count the number of satisfiable instances to the Boolean formula corresponding to the symmetric difference of the target Boolean formula and the formula computed by the current crossbar design. Lines 8 through 11 are a succinct description of the simulated annealing algorithm. New crossbar designs are always accepted if they are better than the existing crossbar design. New crossbar designs that are worse than the existing crossbar design are accepted with a probability that is a function of both the quality of the designs and the current temperature of the simulated annealing algorithm. At every iteration of the loop, the temperature of the simulated annealing algorithm is slightly lowered in Line 11 of the pseudocode. In our approach, we perturbed one memristor in the design at any specific time and the probability of the memristor being perturbed was proportional to the number of times the variable corresponding to the memristor occurs in the Boolean decision diagram representation of the symmetric difference between the Boolean formula and the design. If a variable did not occur in the symmetric difference, it was not perturbed as the remaining error in the design is not related to this variable. When the number of satisfiable instances for the symmetric difference becomes zero, our algorithm stops and reports the synthesized crossbar design.

IV. Experimental Results

We used our approach to synthesize compact adders of various small bit-widths *that are optimized for area*. Since crossbars are regular structures, it is enough to minimize the product of the number of rows and columns in the crossbar. The switching energies and the switching delay of memristors have been obtained from [16]. The switching delay is 85 picoseconds and the average power consumption for switching is $30\mu W$.

Fig. 2. The crossbar design on the left computes the most significant bit of the sum of the two bit-vectors $a[0:2]$ and $b[0:2]$. The crossbar design on the right does the same for 4-bit inputs $a[0:3]$ and $b[0:3]$. Each memristor in the crossbar is labelled with the value that should be stored in the memristor – either a turned-off memristor (black), a bit of the input a (colored light blue), a bit of the input b (colored light brown), negation of a bit of the input a (colored deep blue) or negation of a bit of the input b (colored deep brown).

Figure 1 computes the most significant bit (MSB) of the sum of two 2-bit inputs. It also shows three flows of current from the lowest horizontal nanowire to the topmost horizontal nanowire. The flow of current shown using green dashed line computes the Boolean formula $b[1]\neg a[0]\neg a[1]$ and produces a flow when the least significant bits of the inputs are 0 and the MSB of the second input is 1. Similarly, the red dotted flow occurs when the least significant bits (LSB) of the inputs are 0 and the MSB of the first input is 1. The brown dashed-dotted line in Figure 1 has a flow of current if and only if the two MSBs of the two inputs are 0 but both their LSBs are 1.

Figure 2 shows two crossbar designs: the one of the left computes the MSB of the sum of two 3-bit inputs while the one on the right does the same for 4-bit inputs.

In Table I (see page 1), we compare the size of the crossbar generated by our method to the number of memristors used by two recently proposed methods: BDD based method [5] and AIG based technique [3]. We see that our approach is able to synthesize crossbars that *need as many* 6.3 *times fewer memristors* that competing approaches. We note that the technique based on first-order decision procedures has not been reported to synthesize crossbars of size more than one bit [6].

TABLE II
Computation delay (picoseconds) to compute the MSB of n-bit addition for area-optimized crossbars.

#bits / input	BDD [5]	MIG -IMP [3]	MIG -MAJ [3]	Our Method	Speedup Prior best/ Our method
2	425	3400	1020	340	**125**%
3	765	5100	1530	340	**225**%
4	1020	6800	2040	425	**240**%

TABLE III
POWER (IN μW) REQUIRED TO COMPUTE THE MSB OF BINARY ADDITION FOR AREA-OPTIMIZED CROSSBARS.

#bits / input	BDD [5]	MIG -IMP [3]	MIG -MAJ [3]	Our Method	Power Ratio Prior best/ Our method
2	240	1200	360	300	80%
3	420	1800	540	390	**107.7%**
4	600	2400	720	720	83.3%

In Tables II and III, we compare the speed and energy requirements of our crossbar designs to memristive circuits designed using competing approaches [3], [5]. We recall that a single row or column of memristors in a memristor crossbar can be programmed simultaneously. Our area-optimized circuits *are faster* than all the three competing approaches we investigated. Further, these designs are at least as energy efficient as the circuits presented in two recently published approaches based on majority function computation and implication logic [3].

Figure 3 shows the design of a comparator crossbar for computing whether one input of size 4 bits is at least as large as another input of 4 bits. Using existing approaches, we obtain a crossbar of size 24×16 for solving this problems. Hence, our current approach produces a crossbar that is 12 times *smaller* in area than the existing state-of-the-art for 4-bit comparison.

Fig. 3. A 4-bit comparator implemented on a 8×4 crossbar.

V. CONCLUSIONS & FUTURE WORK

We have introduced a new algorithm for synthesizing compact memristor crossbars that can evaluate Boolean formula and demonstrated it on interesting examples, such as a 4-bit adder and a 4-bit comparator. Our adder circuits are up to 3 times faster than existing approaches and require up to 6.3 times less area. Their energy performance is at least as good as two recently proposed approaches [3] and within 80% of the BDD-based approach [5].

Several interesting directions for future research naturally follow from our observations. In our present work, we have used a uniform distribution to initialize the mapping of the memristors in the crossbar. We envision learning the distribution of memristors across various variables from the crossbars generated *ab initio* and using this distribution to guide our search. We believe that an admissible heuristic for A* optimal search algorithms can be developed by using the information contained in the symmetric difference of the crossbar and the Boolean formula being implemented. We are also investigating the application of variants of this algorithm to large circuits, such as 64-bit adders, by using sciduction [17] on crossbars for adders with smaller bit-widths.

ACKNOWLEDGMENT

The authors would like to thank the US Air Force for support provided through the AFOSR Young Investigator Award to Sumit Jha. The authors acknowledge support from the National Science Foundation Software & Hardware Foundations #1438989 and Exploiting Parallelism & Scalability #1422257 projects. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-16-1-0255.

REFERENCES

[1] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016, US Patent 9,319,047.

[2] S. Chakraborti, P. V. Chowdhary, K. Datta, and I. Sengupta, "BDD based synthesis of boolean functions using memristors," in *2014 9th International Design and Test Symposium (IDT)*. IEEE, 2014, pp. 136–141.

[3] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 948–953.

[4] A. Velasquez and S. K. Jha, "Automated synthesis of crossbars for nanoscale computing using formal methods," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2015, pp. 130–136.

[5] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, p. in press.

[6] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 1870–1873.

[7] Y. N. Joglekar and S. J. Wolf, "The elusive memristor: properties of basic electrical circuits," *European Journal of Physics*, vol. 30, no. 4, p. 661, 2009.

[8] S. Ghosh, R. V. Joshi, D. Somasekhar, and X. Li, "Guest editorial emerging memories?technology, architecture and applications (first issue)," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 105–108, 2016.

[9] F. Merrikh-Bayat and S. B. Shouraki, "Memristor-based circuits for performing basic arithmetic operations," *Procedia Computer Science*, vol. 3, pp. 128–132, 2011.

[10] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1327–1332.

[11] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[12] T. S. Lande, *Neuromorphic systems engineering: neural networks in silicon*. Springer Science & Business Media, 1998, vol. 447.

[13] S. Chakraborti, P. V. Chowdhary, K. Datta, and I. Sengupta, "BDD based synthesis of boolean functions using memristors," in *2014 9th International Design and Test Symposium (IDT)*. IEEE, 2014, pp. 136–141.

[14] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[15] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. ios press, 2009, vol. 185.

[16] B. J. Choi, A. C. Torrezan, J. P. Strachan, P. Kotula, A. Lohn, M. J. Marinella, Z. Li, R. S. Williams, and J. J. Yang, "High-speed and low-energy nitride memristors," *Advanced Functional Materials*, 2016.

[17] S. K. Jha, "Towards automated system synthesis using sciduction," Ph.D. dissertation, University of California, Berkeley, 2011.