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

Dwaipayan Chakraborty  Sumit Kumar Jha

Computer Science Department
University of Central Florida
Orlando, Florida, USA

May 31, 2017

Supported by the National Science Foundation and Air Force Young Investigator Award to Sumit Jha
Table of contents

1 Introduction
   • Motivation
   • Problem Definition

2 Approach
   • Related Work
   • Our Approach

3 Conclusions
   • Experimental Results
   • Conclusions & Future Work
Memristor Crossbars (xbars)

Our computational fabric

Memristor crossbars can make good memory devices:
- Non-volatile
- High-density
- High-speed switching
Mapping C Programs to Memristor Crossbars - I

Our long-term goal

Write a program in a suitable subset of the C language.
Our long-term goal

Write a program in a suitable subset of the C language.

Press a button and obtain the memristor xbar design that implements the C program.
Computing logical “AND” in a constrained topology

Difficult to implement logic on individual memristors

- Feed a current into the top-right nanowire
Computing logical “AND” in a constrained topology

- Bottom-left nanowire has a flow (shown in orange) if and only if a is true.
Computing logical “AND” in a constrained topology

- Top-left nanowire has a flow (shown in yellow) if and only if \((a \text{ and } b)\) is true.
Any Boolean formula can be implemented using such flow-based computing in nanoscale memristive crossbars\(^1\)

Given a Boolean formula $\phi$ over variables $v_1, v_2 \ldots v_n$, design a crossbar such that the topmost horizontal nanowire $r_n$ has a flow if and only if the formula is true for a given assignment $\nu'$ of values to the variables.
Problem Definition - II

Design Crossbar for Evaluating a Boolean Formula

- Given a Boolean formula $\phi$ over variables $v_1, v_2 \ldots v_n$, design a crossbar such that the topmost horizontal nanowire $r_n$ has a flow if and only if the formula is true for a given assignment $V$ of values to the variables.

- Design of the crossbar is a mapping $D$ from memristors $m_{ij}$ to values of variables, their negations, true or false e.g. $D(m_{00}) = V(a)$, $D(m_{03}) = V(b)$. 
Design Space $= \mathcal{O}(2^{\#\text{variables}} + 2^{\#\text{xbarsize}})$

**Large Design Space**
- 4 variables, $4 \times 3$ memristor crossbar
- Each memristor has 10 possibilities
- $\# \text{ Designs} = 10^{(4 \times 3)} = 1 \text{ trillion}$

---

### Synthesis using Decision Procedures - ISCAS 2016

#### Fabricated & Tested in the Laboratory

- Correct response on all 8 inputs (Nathan Cady, SUNY CNSE, Albany - ISCAS 2016)

#### Poor Scalability

- Does not scale to larger designs e.g. 4-bit adders.
Synthesis using BDDs - DATE 2017

Good Scalability
- Scales to large circuits, such as 64-bit adders.

Not as compact
- Circuits are not as compact as those designed by decision procedures.

<table>
<thead>
<tr>
<th>x1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1</td>
<td>1</td>
</tr>
<tr>
<td>x0</td>
<td>0</td>
</tr>
<tr>
<td>y0</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Carry-bit of 2-bit adder
Simple rules of exploration

- Always move from design 1 to design 2 as fitness is improved.
- Probabilistically move from design 2 to design 3, even though its fitness is lower.
Fitness Function

Fitness: -(Design $D$ $\oplus$ Target Formula $\phi$ )

- Expensive to compute the truth table of Design $D$
- $\oplus$ Target Formula $\phi$
  - Logical simulation of flows inside the crossbar
  - For every input combination
Fitness using Symbolic Model Counting - I

Fitness using BDDs
- Compute
  - BDD for target formula
  - BDD for candidate design
  - Symmetric difference of the two BDDs
- Runtime linear in the size of BDDs.
Fitness using Approximate Model Counting - I

N(f)

N(f|_{v=\text{True}})

Easier to count

Satisfying instance of Boolean formula $f$

Satisfying instance of a simpler formula $f|_{v=\text{True}}$

Simplify recursively

- Instead of counting solutions to a formula $f$,
- Count solutions to a simpler formula $f|_{v=\text{True}}$
Build back solutions for larger functions approximately

- Knowing the number of solutions to $f|_{v=\text{True}}$
- Uniformly sample the space of solutions of $f$ to estimate $t$
### MSB of a 2-bit Adder

By comparison, a BDD-based approach needs 8 rows and 5 columns.

**Comparison to DATE 2017**

<table>
<thead>
<tr>
<th>a[0]</th>
<th>b[0]</th>
<th>¬a[1]</th>
<th>¬b[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬a[0]</td>
<td>b[0]</td>
<td>¬b[0]</td>
<td></td>
</tr>
<tr>
<td>¬a[0]</td>
<td>a[0]</td>
<td>¬a[0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>¬b[1]</td>
<td>a[1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a[1]</td>
<td>¬a[1]</td>
<td></td>
</tr>
</tbody>
</table>

- a[0] b[0] ¬a[1] ¬b[1]
- b[1] ¬a[0] ¬a[1]
- a(1) ¬b[1] ¬a[0]
MSB of 3-bit and 4-bit Adders

Comparison to DATE 2017

3-bit adder needs 15 rows and 9 columns.
4-bit adder needs 21 rows and 12 columns.
### 4-bit Comparator

<table>
<thead>
<tr>
<th></th>
<th>(a[3])</th>
<th>(\neg a[3])</th>
<th>(a[0])</th>
<th>(\neg a[0])</th>
<th>(a[1])</th>
<th>(\neg a[1])</th>
<th>(a[2])</th>
<th>(\neg a[2])</th>
<th>(a[3])</th>
<th>(\neg a[3])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a[0])</td>
<td>(a[0])</td>
<td>OFF</td>
<td></td>
<td>OFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg a[0])</td>
<td>(\neg b[0])</td>
<td>OFF</td>
<td></td>
<td>OFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a[1])</td>
<td>(\neg b[1])</td>
<td>OFF</td>
<td></td>
<td>(a[1])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg b[1])</td>
<td>OFF</td>
<td></td>
<td></td>
<td>ON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a[2])</td>
<td>(\neg b[2])</td>
<td>(\neg b[2])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg a[2])</td>
<td>OFF</td>
<td></td>
<td>(a[2])</td>
<td>(a[2])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg a[3])</td>
<td>(\neg b[3])</td>
<td>(a[3])</td>
<td>(a[3])</td>
<td></td>
<td>OFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Comparison to DATE 2017**

4-bit comparator needs 24 rows and 16 columns.
## n-bit Adder – Power Consumption (in $\mu W$)

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>BDD -IMP</th>
<th>MIG -MAJ</th>
<th>Our Method</th>
<th>Power Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>240</td>
<td>1200</td>
<td>360</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>420</td>
<td>1800</td>
<td>540</td>
<td>390</td>
</tr>
<tr>
<td>4</td>
<td>600</td>
<td>2400</td>
<td>720</td>
<td>720</td>
</tr>
</tbody>
</table>

Why this performance benefit?

Other methods require repeated switchings of memristors to implement a logic - thereby requiring more power. Smaller crossbars than BDD based approach.
### n-bit Adder – Computational Delay (in ps)

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>BDD -IMP</th>
<th>MIG -IMP</th>
<th>MIG -MAJ</th>
<th>Our Method</th>
<th>Speedup Prior best/Our method</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>425</td>
<td>3400</td>
<td>1020</td>
<td>340</td>
<td>125%</td>
</tr>
<tr>
<td>3</td>
<td>765</td>
<td>5100</td>
<td>1530</td>
<td>340</td>
<td>225%</td>
</tr>
<tr>
<td>4</td>
<td>1020</td>
<td>6800</td>
<td>2040</td>
<td>425</td>
<td>240%</td>
</tr>
</tbody>
</table>

**Why this low delay?**

No repeated switching of the same memristor or micro-operations. Smaller crossbars than BDD based approach.
### Comparison to DATE 2017*

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>DATE 2017* (multiple xbars)</th>
<th>DATE 2017* (single xbars)</th>
<th>Our work (upper bound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100</td>
<td>212</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>432</td>
<td>606</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>1,408</td>
<td>1,386</td>
<td>160</td>
</tr>
</tbody>
</table>

Looks promising for us

4-bit adders: Our approach has an area less than $4 \times 40 = 160$ compared to 1408 and 1386.

*XbarGen: a tool for design space exploration of memristor based crossbar architectures, DATE 2017.*
Conclusions

What has been achieved so far?

- Synthesis of interesting circuits like 64-bit adders using memristors arranged in a crossbar topology.
- Fabrication and test of 1-bit full adder.
- Compact crossbars for formula like 4-bit adders.

What remains? Synthesis of . . .

- Crossbars for exponential BDDs e.g. multipliers.
- Compact crossbars for circuits e.g. 64-bit adders.
- Multi-valued logic and stochastic crossbar circuits.

Come to our talk at IEEE NANO in Pittsburgh this July.
Some more interesting questions?

- Are crossbars more succinct than BDDs?
- How to compose two crossbars for Boolean formula $\phi_1$ and $\phi_2$ to obtain a crossbar for formula $\phi_1 \land \phi_2$?
- How to design crossbars that are robust to single bit-flips?
- Feynman Grand Prize: Build a 8-bit adder in a cube of side 50nm each.