Automated synthesis of compact crossbars for sneak-path based in-memory computing

Dwaipayan Chakraborty   Sumit Kumar Jha

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

March 29, 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)

Storage
Memristor crossbars can make good storage devices:
- Non-volatile
- High-density
- High-speed switching
Computing constrained by topology

Difficult to implement logic on individual memristors

- Circuit layout should be embedded on the crossbar
Crossbar Computing - Implementing AND (I)

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.
Crossbar Computing - Implementing AND (IV)

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 $\mathcal{V}$ of values to the variables.
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 $\mathcal{V}$ of values to the variables.

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

Large Design Space
- 4 variables, \( 4 \times 3 \) memristor crossbar
- Each memristor has 10 possibilities
- \# Designs = \( 10^{(4\times3)} = 1 \ \text{trillion} \)

---

Synthesis using Decision Procedures

\[ \exists \text{ design}, \forall \text{ inputs s.t. flow} \iff \text{ formula} \]

\[
\forall i \in \{2, \ldots, n\}, r_i^{(t+1)} \iff (r_i^{(t)} \lor \bigvee_{1 \leq j \leq n} (m_{ij} \land c_j^{(t)}))
\]

\[
\forall j \in \{1, \ldots, m\}, c_j^{(t+1)} \iff (c_j^{(t)} \lor \bigvee_{1 \leq i \leq l} (m_{ij} \land r_i^{(t)}))
\]
Fabricated & Tested in the Laboratory

- Correct response on all 8 inputs (SUNY CNSE, Albany)

Poor Scalability

- Does not scale to larger nanoscale crossbars
Step 1: Boolean Formula to BDDs

<table>
<thead>
<tr>
<th>a0</th>
<th>a1</th>
<th>b0</th>
<th>b1</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Decision Diagrams
- Canonical representation
- Often succinct; worst-case exponential
- Good tool support: BuDDy, CUDD, Python DD & others

Jha@cs.ucf.edu
Automated Synthesis + In-Memory Computing
Step 2: Laying out BDDs on the Crossbar (I)

Boolean Decision Diagrams
- Can be imagined to perform flow-based computing
- Nodes map to nanowires, edges map to memristors?
Step 2: Laying out BDDs on the Crossbar (II)

Boolean Decision Diagrams

- Can be imagined to perform flow-based computing
Let us try

- Mapping nodes to nanowires
- Mapping edges to memristors
Step 2: Laying out BDDs on the Crossbar (IV)

But, we get stuck
- Orange wire should be connected to a horizontal nanowire x0
- and a vertical nanowire y0
Step 2: Our Approach for Laying out BDDs (I)

A single level of a Binary Decision Diagram and the children of the nodes at this level. The logical value of the children coupled with the logical value of the variable labeling this node produces the logical value computed by the nodes at this level.
A single level of a Binary Decision Diagram and the children of the nodes at this level. The logical value of the children coupled with the logical value of the variable labeling this node produces the logical value computed by the nodes at this level.

\[ x_0 = x_0^r x + x_0^l \neg x \]
Subgraph $G_{\Phi}[i : j]$ can be mapped to a crossbar with the inputs (corresponding to level $j$ of the ROBDD) being fed along the lower rows and the outputs (corresponding to level $i$) leaving the crossbar along the higher rows.
Subgraph $G_\phi[i : j]$ can be mapped to a crossbar with the inputs (corresponding to level $j$ of the ROBDD) being fed along the lower rows and the outputs (corresponding to level $i$) leaving the crossbar along the higher rows.
Carry bit of a 2-bit Adder (I)
Carry bit of a 2-bit Adder (II)
Carry bit of a 2-bit Adder (III)
Carry bit of a 2-bit Adder (III)
### Power Consumption (in $\mu$W)

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>BDD</th>
<th>MIG -IMP</th>
<th>MIG -MAJ</th>
<th>Our Method</th>
<th>Prior best/Our method</th>
</tr>
</thead>
<tbody>
<tr>
<td>5xp1</td>
<td>2,190</td>
<td>2,970</td>
<td>1,080</td>
<td>393</td>
<td>274.8%</td>
</tr>
<tr>
<td>9sym_d</td>
<td>1,860</td>
<td>5,250</td>
<td>1,800</td>
<td>1,027</td>
<td>175.3%</td>
</tr>
<tr>
<td>misex3</td>
<td>5,550</td>
<td>4,950</td>
<td>2,010</td>
<td>1,901</td>
<td>105.7%</td>
</tr>
<tr>
<td>sym10_d</td>
<td>2,100</td>
<td>5,610</td>
<td>2,160</td>
<td>1,479</td>
<td>146.0%</td>
</tr>
<tr>
<td>clip</td>
<td>2,670</td>
<td>3,300</td>
<td>1,200</td>
<td>605</td>
<td>198.3%</td>
</tr>
</tbody>
</table>

**Why this performance benefit?**

Other methods require **repeated switchings** of memristors to implement a logic - thereby requiring more power. Our method is constrained by the crossbar topology.
Computational Delay (in picoseconds)

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>BDD</th>
<th>MIG -IMP</th>
<th>MIG -MAJ</th>
<th>Our Method</th>
<th>Prior best/Our method</th>
</tr>
</thead>
<tbody>
<tr>
<td>5xp1</td>
<td>6,205</td>
<td>8,415</td>
<td>3,060</td>
<td>1,109</td>
<td>275.9%</td>
</tr>
<tr>
<td>9sym_d</td>
<td>5,270</td>
<td>14,875</td>
<td>5,100</td>
<td>2,898</td>
<td>200.1%</td>
</tr>
<tr>
<td>misex3</td>
<td>15,725</td>
<td>14,025</td>
<td>5,695</td>
<td>5,369</td>
<td>106.1%</td>
</tr>
<tr>
<td>sym10_d</td>
<td>5,950</td>
<td>15,895</td>
<td>6,120</td>
<td>4,192</td>
<td>145.9%</td>
</tr>
<tr>
<td>clip</td>
<td>7,585</td>
<td>9,350</td>
<td>3,400</td>
<td>1,706</td>
<td>199.3%</td>
</tr>
</tbody>
</table>

**Why this low delay?**

No repeated switching of the same memristor or micro-operations.

Our method is constrained by the crossbar topology.
### Comparison to DATE 2017*

<table>
<thead>
<tr>
<th>#bits / input</th>
<th>DATE 2017* (lower bound)</th>
<th>DATE 2017* (lower bound)</th>
<th>Our work (MSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100</td>
<td>212</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>432</td>
<td>606</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>1,408</td>
<td>1,386</td>
<td>252</td>
</tr>
<tr>
<td>8</td>
<td>57,592</td>
<td>8,954</td>
<td>544</td>
</tr>
<tr>
<td>16</td>
<td>1,430,800</td>
<td>25,366</td>
<td>2,112</td>
</tr>
<tr>
<td>32</td>
<td>-</td>
<td>-</td>
<td>8,320</td>
</tr>
</tbody>
</table>

**Looks promising for us**

4-bit adders: Our approach has an area of $252 + 135 + 40 + 20 = 447$ 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 non-trivial circuits like 64-bit adders using memristors arranged in a crossbar topology.
- Fabrication of 1-bit full adder tested manually on its 8 inputs.

What remains?

- Fabrication of 8-bit full adder and (robotic?) testing of its $2^{16}$ inputs.
- How do you test a 128-bit adder in this framework?
- Resilience to faults (known and unknown).
- Circuits with exponential BDDs e.g. multipliers.

Jha@cs.ucf.edu
Automated Synthesis + In-Memory Computing