Technology-aware Logic Synthesis For ReRAM Based In-memory Computing

Debjyoti Bhattacharjee¹, Luca Amaru² and Anupam Chattopadhyay¹

debjyoti001@ntu.edu.sg

1. School of Computer Science and Engineering, Nanyang Technological University, Singapore
2. Synopsys Inc, US
Outline

Background

Technology Aware Logic Synthesis

Results
1. The programmable logic-in-memory (PLiM) computer, Galliardon et al., DATE 2016
2. ReVAMP: ReRAM-based VLIW Architecture for in-Memory computing, Bhattacharjee et al., DATE 2017
3. Logic design within memristive memories using memristor-aided LoGIC (MAGIC), Talati et al., IEEE Trans 2016
General Purpose Programmable Architectures

Primitive Boolean functions natively realized vary.

ReRAM devices arranged as a crossbar used for computation

Crossbar constraints differ across architectures


DATE 2018, Dresden 22-03-2018
ReVAMP
A VLIW-like Architecture using ReRAM crossbar memory

1. ReVAMP: ReRAM based VLIW Architecture for in-Memory computing, Bhattacharjee et al., DATE 2017

DATE 2018, Dresden 22-03-2018
ReVAMP

A VLIW-like Architecture using ReRAM crossbar memory

➢ All the devices are reset to 0 initially
➢ Values are applied via a wordline and the bitlines to perform computation
➢ This is specified using Apply instruction

Apply w ws wb ($v \text{val}_{63}$)($v \text{val}_{62}$) ... ($v \text{val}_0$)
ReVAMP
A VLIW-like Architecture using ReRAM crossbar memory

➢ The value stored in a word has to be read out for meaningful logical operations
➢ This is specified using Read instruction ➔ Available in DMR

Read w
Design specified in HDL [Verilog, blif, etc]

ABC, Cirkit, etc [Generates logic network]

Support for multiple architectures are already available

Technology aware optimization is missing!
Outline

Background

Technology Aware Logic Synthesis

Results
Technology aware Logic Synthesis

MIG  Algebraic Rewriting  Boolean IPM  Majority Refactoring
Technology aware Logic Synthesis
Technology aware Logic Synthesis

The axiomatic system for the MIG Boolean algebra, referred to as $\Omega$

\[
\begin{align*}
\text{Commutativity} & : \Omega.C \\
M(x, y, z) &= M(y, x, z) = M(z, y, x) \\
\text{Majority} & : \Omega.M \\
if(x = y) : M(x, y, z) &= x = y \\
if(x = \overline{y}) : M(x, y, z) &= z \\
\text{Associativity} & : \Omega.A \\
M(x, u, M(y, u, z)) &= M(z, u, M(y, u, x)) \\
\text{Distributivity} & : \Omega.D \\
M(x, y, M(u, v, z)) &= M(M(x, y, u), M(x, y, v), z) \\
\text{Inverter Propagation} & : \Omega.I \\
\overline{M}(x, y, z) &= M(\overline{x}, \overline{y}, \overline{z})
\end{align*}
\]

“Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization,” Amaru et al., DAC 2014
Technology aware Logic Synthesis

- MIGs ➔ hierarchical majority voting systems.
- Majority voting ➔ Capable of correcting different types of bit-errors.
- Error masking property can be exploited for logic optimization.
- Purposely introduce logic errors.
- Choose inputs with the highest dictatorship for inserting errors.
- Dictatorship: Ratio of input patterns over the total \(2^n\) for which the output assumes the same value than the selected input.

\[
f = (a + b)c \\
\text{Dictatorship}(a) = 5/8 \\
\text{Dictatorship}(b) = 5/8 \\
\text{Dictatorship}(c) = 7/8
\]
➢ Select a subset of the primary inputs with highest dictatorship.
➢ For each selected input, we determine a condition that causes an error.
➢ These errors have to be orthogonal.

➢ $x, y$ and $z$ can be partitioned using the following partition
   ➢ $x \neq y; x = y = z; x = y = z'$
   ➢ Corresponding errors are
   ➢ $A : x = y; B : z = \overline{y}$ when $x = y; C : z = y$

“Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization,” Amaru et al., DAC 2014
Input Partitioning Methods

➢ Consider \( f = M(x, M(x, y', z), M(x', y, z)) \)

➢ For error A: \( x = y \),
  \[
  f_A = M(x, M(y, y', z), M(x', x, z)) = M(x, z, z) = z
  \]

➢ For error B: \( z = y' \),
  \[
  f_B = M(x, M(x, y', y'), M(x', y, y')) = M(x, y', x') = y'
  \]

➢ For error C: \( z = y \),
  \[
  f_C = M(x, M(x, z', z), M(x', z, z)) = M(x, x, z) = x
  \]

➢ Thus, \( f = M(f_A, f_B, f_C) = M(z, y', x') \)
  ➢ Two levels collapse into a single level
  ➢ Node count reduces by 2
Technology aware Logic Synthesis

- Cone of logic $\rightarrow$ canonical logic representation (BDD, Truth Table)
- Canonical logic representation $\rightarrow$ a new local MIG (majority decomposition)
- Compute cost of the new local MIG
  - If $\text{cost(}\text{local MIG}) < \text{cost(}\text{original logic cone})$: Local MIG is imported back to the network
- Repeat for every node of the MIG in topological order
  - Stop if no more improvement or a computation limit is reached
Technology aware Logic Synthesis

Optimization goals

➢ #nodes indicates the number of computations needed.
   ➢ reduction of number nodes in MIG is primary objective
➢ For ReRAM devices arranged as a crossbar,
   ➢ All the operations in a level of the logic networks might not be computed in parallel
➢ Depth of MIG does not directly translate to delay
➢ Rationale for crossbar-aware optimization:
   ➢ Enforce logic sharing in the MIG
   ➢ Share non-inverted edges,
     ➢ if multiple inverted-edges are shared, propagate the inverts above.
Outline

Background

Technology Aware Logic Synthesis

Results
Results

Performance on HARD EPFL benchmarks

- Depth-optimized hard EPFL benchmarks, available as MIG
- Word length of crossbar = 16
- Default technology mapping flow for ReVAMP used.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>#Words</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder</td>
<td>3369</td>
<td>9333</td>
<td>255</td>
<td>12597</td>
</tr>
<tr>
<td>adder(opti)</td>
<td>2811</td>
<td>7662</td>
<td>176</td>
<td>7603</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>adder</th>
<th>div</th>
<th>log2</th>
<th>max</th>
<th>mult</th>
<th>sin</th>
<th>sqrt</th>
<th>square</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>3369</td>
<td>76005</td>
<td>37650</td>
<td>7846</td>
<td>42194</td>
<td>7946</td>
<td>52538</td>
<td>19395</td>
</tr>
<tr>
<td>Edges</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Words</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Improvement in Technology Mapping (in percentage)
Results

Performance on HARD EPFL benchmarks

➢ On average,
  ➢ Reduction #nodes = 10.8%  
    (maximum reduction 16.56%)
  ➢ Reduction in overall delay = 16.67%  
    (maximum reduction of 39.64%)
  ➢ For all the benchmarks, > 99% device utilization achieved by the technology mapping algorithm.
Results

Impact of word length on Synthesis optimizations

Improvement in Delay

Improvement in #Words
Conclusion

- Proposed a novel crossbar-aware MIG optimization automation flow
- Integrated into existing crossbar-aware technology mapping flow
- Demonstrated performance benefits over large benchmarks
  - Significant reduction in delay and number of words required for mapping
  - Enabled improved performance with increase in word length
Technology-aware Logic Synthesis For ReRAM Based In-memory Computing

Debjyoti Bhattacharjee¹, Luca Amaru² and Anupam Chattopadhyay¹

debjyoti001@ntu.edu.sg

1. School of Computer Science and Engineering, Nanyang Technological University, Singapore
2. Synopsys Inc, US