Automating Logic Rectification by Approximate SPFDs

Yu-Shen Yang
Andreas Veneris
Subarna Sinha
Robert K. Brayton

University of Toronto
Synopsys Inc.
UC Berkeley
Outline

• Introduction
• SPFDS
• Approximating SPFDS
• Automating Rectification
• Experiments
• Conclusion
Introduction

• Synthesized designs are often readjusted to achieve different goals.
  – Debugging
  – Engineering Change
  – Rewiring

• *Logic transformation* is required
  – Restructures the design locally
  – Minimizes modifications
  – Preserves the previous engineering efforts

Example: Rewiring
Introduction

• Most logic transformation approaches use *dictionary models*.
  – Predetermined
  – Simple transformations

• Problem
  – Not adequate for complex transformations
  – Low hit ratio

• Algorithmic transformation (dynamic, non-predetermined) is desired → **SPFDs**

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1908_s</td>
<td>27%</td>
</tr>
<tr>
<td>C2670_s</td>
<td>11%</td>
</tr>
<tr>
<td>C5315_s</td>
<td>25%</td>
</tr>
<tr>
<td>C3540_c</td>
<td>6%</td>
</tr>
<tr>
<td>C5315_c</td>
<td>16%</td>
</tr>
<tr>
<td>C7552_c</td>
<td>19%</td>
</tr>
</tbody>
</table>
Introduction

• SPFDs
  – New representation of Boolean functions [Yamashita et al., ICCAD’96]
  – Ideal for resynthesis. [Sinha, Brayton, IWLS’98]
  – Works on a large input space → Memory/runtime intensive

• Contribution I:
  – A simulation-based method to approximate SPFDs (aSPFD)
  – Reduce the complexity of SPFDs

• Contribution II:
  – Automate logic transformations using aSPFD
  – Increase hit ratio up to 100%
  – SAT-based and greedy approaches
Outline

- Introduction
- **SPFDs**
- Approximating SPFDs
- Automating Rectification
- Experiments
- Conclusion
SPFDs

- **Sets of Pairs of Functions to be Distinguished** [Yamashita et al., ICCAD’96]
  - Originally for applications to FPGAs

- An alternative way to express *functional flexibility*
  - *Good for resynthesis*

- Can be presented as a graph [Sinha, Brayton IWLS’98]
SPFDs

SPFD of a function, \( f \)

- Each minterm of \( f \) is a vertex
- An edge exists between \((m_1, m_2)\) if \( f(m_1) \neq f(m_2) \)
- A vertex with no edge is a don't care
- Values of the vertexes are not specified
  - Many functions can have the same SPFD
Outline

• Introduction
• SPFDDs
• Approximating SPFDDs
• Automating Rectification
• Experiments
• Conclusion
Approximating SPFDs

- **Previous methods**
  - Formulate by BDDs or SAT
  - Analyze the entire set of PI minterms
  - Memory/runtime expensive

- **Approximate SPFDs (aSPFD)**
  - A subset of PI minterms
  - Constructed by simulation vectors
  - Less expensive to manipulate and compute
Approximating SPFDs

• Which minterms should be picked?

• Logic rectification can be viewed in the debugging context [Smith et al., TCAD’05 ]
  – error/correction operations
  – Different errors require different sets of minterms

• Use simulation vectors to represent the behavior of the error
  – Help to select important minterms
Approximating SPFDs

Generate aSPFD of candidate, \( n_{err} \)

1. Simulate good / bad circuits

2. Collect \( V^e, V^c \)

3. Find
   \( \text{on}(V^e), \text{on}(V^c), \text{off}(V^e), \text{off}(V^c) \)

4. Add edges
   \( \text{on}(V^c) \times \text{on}(V^e), \text{off}(V^c) \times \text{off}(V^e) \)

\( \text{on}(V)/\text{off}(V) \) returns vectors that \( n_{err} = 1/0 \)
Approximating SPFDs

Example:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
<th>( z_{\text{error}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
<td>1</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>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{On}(V_c) = \{ 010, 101, 111 \} \]
\[ \text{Off}(V_c) = \{ 100 \} \]
\[ \text{On}(V_e) = \{ \} \]
\[ \text{Off}(V_e) = \{ 110 \} \]

Goal: remove \( e \) ---- \( z \)

\[ \text{On}(V_c) = \{ 010, 101, 111 \} \]
\[ \text{Off}(V_c) = \{ 100 \} \]
\[ \text{On}(V_e) = \{ \} \]
\[ \text{Off}(V_e) = \{ 110 \} \]

\( V_c = \{ 010, 100, 101, 111 \} \)
\( V_e = \{ 110 \} \)
Outline

• Introduction
• SPFDs
• Approximating SPFDs
• **Automating Rectification**
• Experiments
• Conclusion
Automating Rectification

- **Property:** SPFD of a node must be the sub-graph of the union of SPFDs of its fan-ins
  \[ SPFD_n \subseteq SPFD_a \cup SPFD_b \]

- The aSPFD contains edges not belong to the union
  - Missing fan-ins

- Rectify the design by adding fan-ins
  - Covering the extra edges
  - Resynthesis the candidate
    [Cong et al., FPGA’02]

This work
Automating Rectification

Example:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
<th>z_{\text{error}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
<td>1</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>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Goal: remove e → z

\[ \text{aSPFD}_z \]

\[ \text{aSPFD}_b \]
Automating Rectification

Two approaches for searching qualified wires

• SAT-based approach
  – Variables:
    • Represent wires in the design
  – Covering Clauses:
    • Generated for each edge
    • Consist of wires that can cover the edge
  – Blocking Clauses:
    • Prevent selecting a wire when all of its fan-ins has been selected
  – Returns the optimal solution
  – NPC Problem: may run into the runtime issue
Automating Rectification

Two approaches for searching qualified wires

• Greedy approach
  – Procedure:
    • Each time selecting the wire that can covering most edges
    • Repeat until all edges are covered
  – The result may not be the optimal
  – Runtime efficient
Outline

- Introduction
- SPFDs
- Approximating SPFDs
- Automating Rectification
- Experiments
- Conclusion
Experiments

• Three types of errors
  – **Simple**: single simple error
  – **Medium**: combination of simple errors
  – **Complex**: many errors in the fan-in cone

• The locations are provided by a fast linear-time diagnosis method.

• Pseudo-Boolean constraint SAT solver (MiniSat) is used to return the optimal solution [Eén, Sörensson, JSAT’06].
## Experiments

<table>
<thead>
<tr>
<th>circuit</th>
<th>error loc.</th>
<th>Dict. model</th>
<th>aSPFD</th>
<th>Min # wires</th>
<th># of wires</th>
<th>minterm count</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3540_s</td>
<td>7.2</td>
<td>27.8%</td>
<td>86.1%</td>
<td>1.1</td>
<td>1.1</td>
<td>3.7</td>
</tr>
<tr>
<td>C5315_s</td>
<td>6.4</td>
<td>25.0%</td>
<td>100.0%</td>
<td>-</td>
<td>1.9</td>
<td>5.9</td>
</tr>
<tr>
<td>C7552_s</td>
<td>11.8</td>
<td>19.2%</td>
<td>50.0%</td>
<td>-</td>
<td>1.7</td>
<td>4.9</td>
</tr>
<tr>
<td>C3540_m</td>
<td>3.2</td>
<td>25.0%</td>
<td>100.0%</td>
<td>1.6</td>
<td>1.6</td>
<td>4.1</td>
</tr>
<tr>
<td>C5315_m</td>
<td>9.6</td>
<td>2.2%</td>
<td>100.0%</td>
<td>-</td>
<td>2.9</td>
<td>7.4</td>
</tr>
<tr>
<td>C7552_m</td>
<td>8.8</td>
<td>9.1%</td>
<td>90.9%</td>
<td>-</td>
<td>1.9</td>
<td>6.3</td>
</tr>
<tr>
<td>C3540_c</td>
<td>3.0</td>
<td>6.7%</td>
<td>66.7%</td>
<td>3.4</td>
<td>3.6</td>
<td>6.1</td>
</tr>
<tr>
<td>C5315_c</td>
<td>6.4</td>
<td>16.1%</td>
<td>100%</td>
<td>-</td>
<td>2.7</td>
<td>8.4</td>
</tr>
<tr>
<td>C7552_c</td>
<td>20.6</td>
<td>19.1%</td>
<td>50%</td>
<td>-</td>
<td>1.9</td>
<td>5.2</td>
</tr>
</tbody>
</table>
Experiments

• # of selected new wires vs. complexity of errors
Outline

• Introduction
• SPFDS
• Approximating SPFDS
• Automating Rectification
• Experiments
• Conclusion
Conclusion

• A simulation-based method to approximate SPFDs
  – Avoids the memory explosion

• An algorithmic logic rectification using aSPFD
  – Outperform methods with dictionary models
  – SAT-based approach for the optimal solutions
  – Greedy approach for runtime efficiency

• Future works
  – Circuits required rectifications at multiple locations
  – Sequential circuits
Thank you