

Automating Logic Rectification by Approximate SPFDs

Yu-Shen Yang
Andreas Veneris

Subarna Sinha

Robert K. Brayton



University of Toronto



UC Berkeley

Synopsys Inc.

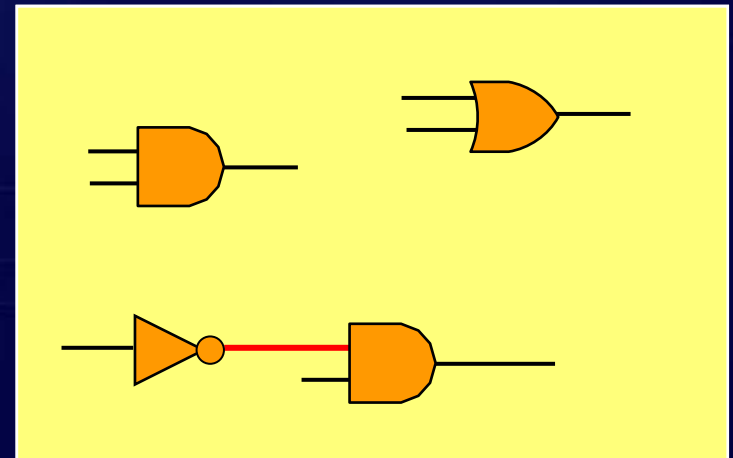
Outline

- **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**

Circuits	Hit ratio
C1908_s	27%
C2670_s	11%
C5315_s	25%
C3540_c	6%
C5315_c	16%
C7552_c	19%

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



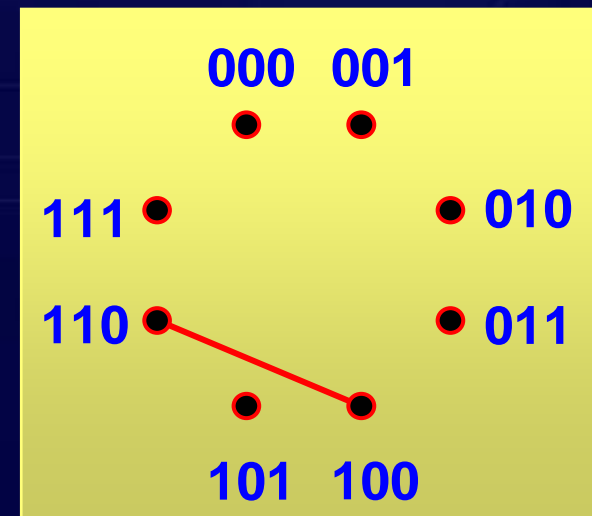
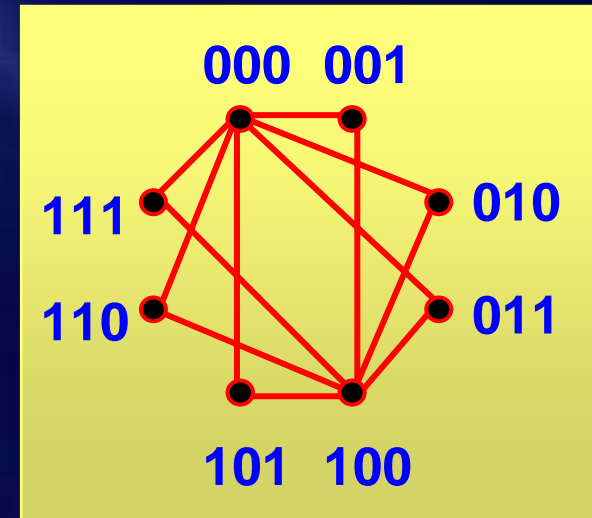
	f_1	f_2	f_3	f_4
000	0	0	1	1
010	1	1	0	0
110	0	1	1	0
111	1	0	0	1

Outline

- Introduction
- SPFDs
- **Approximating SPFDs**
- 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
 $on(V^e), on(V^c),$
 $off(V^e), off(V^c)$



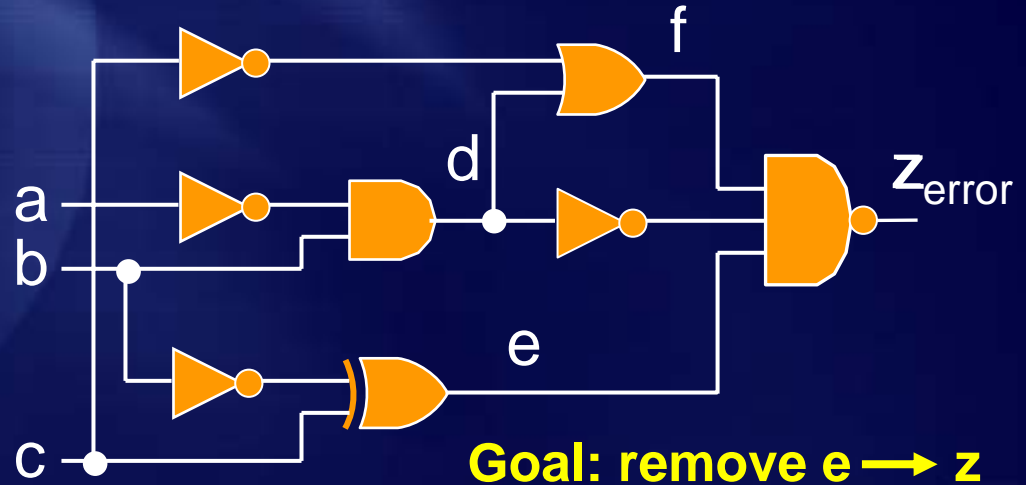
4. Add edges
 $on(V^c) \times on(V^e)$
 $off(V^c) \times off(V^e)$

$on(V)/off(V)$
returns vectors
that $n_{err} = 1/0$

Approximating SPFDs

Example:

a	b	c	z	z _{error}
0	1	0	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1



$$On(V^c) = \{010, 101, 111\}$$

$$V^c = \{010, 100, 101, 111\}$$

$$Off(V^c) = \{100\}$$

$$On(V^e) = \{\}$$

$$V^e = \{110\}$$

$$Off(V^e) = \{110\}$$

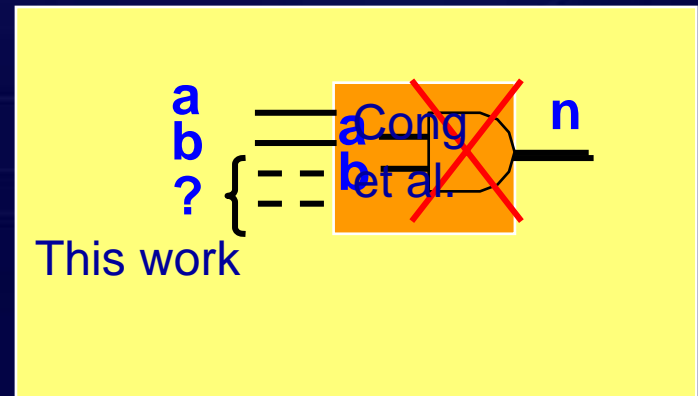


Outline

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

Automating Rectification

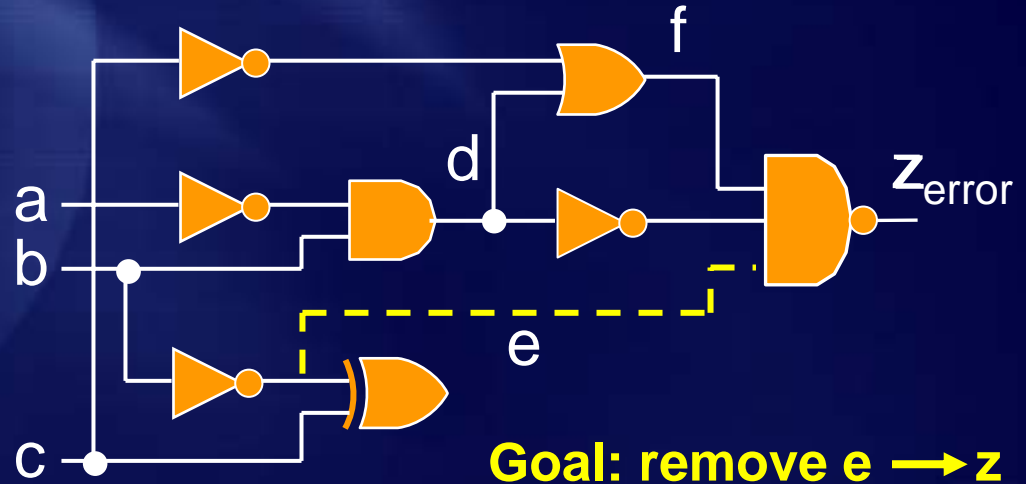
- **Property:** SPFD of a node must be the sub-graph of the union of SPFDs of its fan-ins
 - $SPFD_n \subseteq SPFD_a \cup SPFD_b$
- The aSPFD contains edges not belong to the union
 - **Missing fan-ins**
- Rectify the design by adding fan-ins
 - Covering the extra edges
 - Resynthesis the candidate [Cong et al., FPGA'02]



Automating Rectification

Example:

a	b	c	z	z _{error}
0	1	0	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1



000 001



111

010

110

011



aSPFD_z 101 100

000 001



111

010

110

011



aSPFD_b 101 100

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

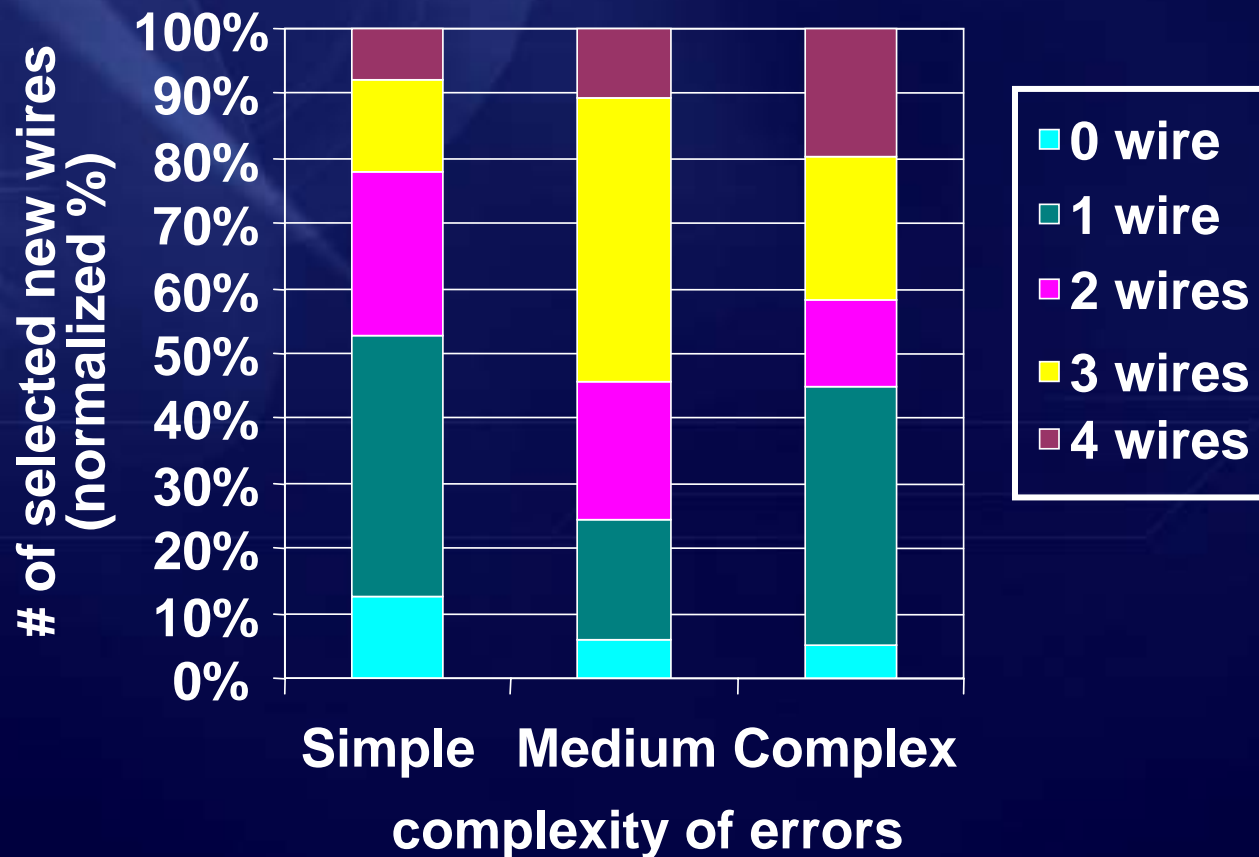
- Three types of errors
 - **Simple**: single simple error
 - **Medium**: combination of simple errors
 - **Complex**: many errors in the fan-in cone
- The locations are provided by a fast linear-time diagnosis method.
- Pseudo-Boolean constraint SAT solver (MiniSat) is used to return the optimal solution [Eén, Sörensson, JSAT'06].

Experiments

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

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