Formal Equivalence Checking

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Dept. of Electrical Engineering
Indian Institute of Technology Bombay, Mumbai
viren@ee.iitb.ac.in

EE 709: Testing & Verification of VLSI Circuits
Lecture – 6 (Jan 17, 2012)
SoC Verification

- System-on-Chip (SOC) design
- Increase of design complexity
- Move to higher levels of abstraction
System-on-Chip (SoC) design

- Specification to architecture and down to implementation
- Behavior (functional) to structure
  - System level: system specification to system architecture
  - RT/IS level: component behavior to component micro-architecture

Specification + constraints
System architecture + estimates
RTL/IS Implementation + results
Verification challenge

Bottlenecks in Design Cycles:
Survey of 545 engineers by EETIMES 2000
### System-level design & verification

Remove as **many bugs** as possible in the **earlier** stages

Do not introduce **new** design errors when refining designs

↓

Formal verification in system-level designs:
Property checking and equivalence checking

<table>
<thead>
<tr>
<th>Level</th>
<th>Bugs Fix Time</th>
<th>Cost Due to Delay/Late Time-to-Market Revenue Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistor</td>
<td>3 minutes</td>
<td>3 minutes</td>
</tr>
<tr>
<td>RTL</td>
<td>3 days</td>
<td>3 days</td>
</tr>
<tr>
<td>System-level</td>
<td>3 weeks</td>
<td>3 weeks</td>
</tr>
</tbody>
</table>
Formal verification

• “Prove” the correctness of designs
  – Both design and spec must be represented with mathematical models
  – Mathematical reasoning
  – Equivalent to “all cases” simulations

• Possible mathematical models
  – Boolean function (Propositional logic)
    • How to represent and manipulate on computers
  – First-order logic
    • Need to represent “high level” designs
  – Higher-order logic
    • Theorem proving = Interactive method

• Front-end is also very important
  – Often, it determines the total performance of the tools
Backgrounds technology in formal verification

- Methods for reasoning about mathematical models
  - Boolean function (Propositional logic)
    - SAT (Satisfiability checker)
    - BDD (Binary Decision Diagrams)
  - First-order logic
    - Logic of uninterpreted functions with equality
  - Higher-order logic
    - Theorem proving = Interactive method
Given two designs, prove that for all possible input stimuli their corresponding outputs are equivalent.
Formal Equivalence Checking

- Scalable
  - Full chip verification possible
  - e.g. Designs of up to several million gates verified in a few hours or minutes with CEC
  - Hierarchical verification deployed

- Automatic
  - CEC: Nearly full automation possible

- High/Full Coverage

CEC Currently most practical and pervasive formal verification technology used in industry
Formal Equivalence Checking

• Equivalence checking can be applied at or across various levels

Diagram:
- System Level
- Gate Level
- Device Level
- RTL

Connections between levels indicate the various levels where equivalence checking can be applied.
Key observation: The circuit being verified usually have a number of internal equivalent functions
Formal Equivalence Checking

 Canonical Forms

\[ f = ab + c \]

\[ F' = (a + c)(b + c) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>
Formal Equivalence Checking

Complexity

- Efficiency of the conversion to canonical form
- Memory requirement
- Efficiency of the comparison of two representation of the canonical form
- Efficiency to generate the counter example in case of a miscompare
Formal Equivalence Checking

- **Satisfiability Formulation**
  - Search for input assignment giving different outputs

- **Branch & Bound**
  - Assign input(s)
  - Propagate forced values
  - Backtrack when cannot succeed

- **Challenge**
  - Must prove all assignments fail
    - Co-NP complete problem
  - Typically explore significant fraction of inputs
  - Exponential time complexity

Jan 17, 2012
Formal Equivalence Checking

- Canonical form representation is only suitable
- DNF and CNF are not suitable
- BDD is most popular canonical form
  - graphical representation of boolean function
Formal Equivalence Checking

- BDD is canonical form of representation
- Shannon's expansion theorem

\[ f(x_1, x_2, \ldots, x_i, \ldots, x_n) = \]
\[ x_i \cdot f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) + \]
\[ x_i' \cdot f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n) \]
Binary Decision Diagram

- Generate Complete Representation of Circuit Function
  - Compact, canonical form

Functions equal if and only if representations identical
- Never enumerate explicit function values
- Exploit structure & regularity of circuit functions
Truth Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Decision Tree

- Vertex represents decision
- Follow green (dashed) line for value 0
- Follow red (solid) line for value 1
- Function value determined by leaf value.
Variable Ordering

- Assign arbitrary total ordering to variables
  - e.g., $x_1 < x_2 < x_3$
- Variables must appear in ascending order along all paths

OK

Not OK

Properties

- No conflicting variable assignments along path
- Simplifies manipulation
Reduction Rule #1

Merge equivalent leaves

```
\[ a \quad a \quad a \]

Jan 17, 2012 EE-709@IITB
Reduction Rule #2

Merge isomorphic nodes

Jan 17, 2012  EE-709@IITB
Reduction Rule #3

Eliminate Redundant Tests

\[ \begin{align*}
    y &\rightarrow x \backslash y \\
    x_1 &\rightarrow x_2 \backslash x_3 \backslash 0 \\
    x_2 &\rightarrow 1 \\
    x_3 &\rightarrow 0 \\
\end{align*} \]
Example OBDD

**Initial Graph**

**Reduced Graph**

- Canonical representation of Boolean function
  - For given variable ordering
  - Two functions equivalent if and only if graphs isomorphic
  - Can be tested in linear time
  - Desirable property: *simplest form is canonical.*
Example Functions

Constants

0  Unique unsatisfiable function
1  Unique tautology

Variable

Treat variable as function

Typical Function

\[(x_1 \lor x_2) \land x_4\]

No vertex labeled \(x_3\)

\(x_3\) independent of \(x_3\)

Many subgraphs shared

Odd Parity

Linear representation

Jan 17, 2012
Representing Circuit Functions

- Functions
  - All outputs of 4-bit adder
  - Functions of data inputs

- Shared Representation
  - Graph with multiple roots
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder

Linear growth

Jan 17, 2012
Effect of Variable Ordering

\((a_1 \land b_1) \lor (a_2 \land b_2) \lor (a_3 \land b_3)\)

**Good Ordering**

- **Linear Growth**
  - \(a_1\) to 0
  - \(b_1\) to 1
  - \(a_2\) to 0
  - \(b_2\) to 1
  - \(a_3\) to 0
  - \(b_3\) to 1

- **Exponential Growth**
  - \(a_1\) to 0
  - \(a_2\) to 1
  - \(a_3\) to 0
  - \(b_1\) to 1
  - \(b_2\) to 0
  - \(b_3\) to 1

**Bad Ordering**

- **Linear Growth**
  - \(a_1\) to 1
  - \(b_1\) to 0
  - \(a_2\) to 1
  - \(b_2\) to 0
  - \(a_3\) to 1
  - \(b_3\) to 0

- **Exponential Growth**
  - \(a_1\) to 1
  - \(a_2\) to 0
  - \(a_3\) to 1
  - \(b_1\) to 0
  - \(b_2\) to 0
  - \(b_3\) to 0
Selecting Good Variable Ordering

• Intractable Problem
  ➢ Even when problem represented as OBDD
    ➢ i.e., to find optimum improvement to current ordering

• Application-Based Heuristics
  ➢ Exploit characteristics of application
  ➢ e.g., Ordering for functions of combinational circuit
    ➢ Traverse circuit graph depth-first from outputs to inputs
    ➢ Assign variables to primary inputs in order encountered
Selecting Good Variable Ordering

Static Ordering
- Fan In Heuristic
- Weight Heuristic

Dynamic Ordering
- Variable Swap
- Window Permutation
- Sifting
Swapping Adjacent Variables

- Localized Effect
  - Add / delete / alter only nodes labeled by swapping variables
  - Do not change any incoming pointers
Dynamic Variable Reordering

Richard Rudell, Synopsys

Periodically Attempt to Improve Ordering for All BDDs

- Part of garbage collection
- Move each variable through ordering to find its best location

Has Proved Very Successful

- Time consuming but effective
- Especially for sequential circuit analysis
Dynamic Reordering By Sifting

- Choose candidate variable
- Try all positions in variable ordering
  - Repeatedly swap with adjacent variable
- Move to best position found
ROBDD sizes & variable ordering

- **Bad News**
  - Finding optimal variable ordering NP-Hard
  - Some functions have exponential BDD size for all orders \( e.g. \) multiplier

- **Good News**
  - Many functions/tasks have reasonable size ROBDDs
  - Algorithms remain practical up to 500,000 node OBDDs
  - Heuristic ordering methods generally satisfactory

- **What works in Practice**
  - Application-specific heuristics \( e.g. \) DFS-based ordering for combinational circuits
  - Dynamic ordering based on variable sifting \( (R. \ Rudell) \)
Thank you