10 Design Verification and Test

Spring 2009
Rajesh Gupta
Outline

- Design Verification
  - Definitions and Terminology
    - Simulation versus Verification
    - Electrical and Timing Verification
    - Formal Verification
- Formal Verification
  - Equivalence Checking
  - Model Checking
- Manufacturing Test: covered later
- Sources:
  - Chapters G, H, Rabaey; Chapter 10, Wolf
  - Sandeep Shukla, Virginia Tech / Mani Srivastava, UCLA
Design Validation versus Manufacturing Test

- Design validation can be done in a number of ways
  - Simulation
  - (Formal) Verification
  - Emulation and Prototyping

- Often combinations of these methods are used at different levels of a design.

- A manufacturing test is designed to certify a manufactured part for use in systems.

- Both validation and test influence the design.
Definitions

› Design synthesis:
- Given an I/O function, develop a procedure to manufacture a device using known materials and processes.

› Simulation/Verification:
- Predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function.
- Two Parts:
  - Functional Verification
  - Performance Verification

› Test:
- A manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect.

[Adapted from VLSI Testing Course by Bushnell/Agrawal at Rutgers]
### Terminology

- **Simulation**
  - whether functional or performance, requires a good understanding of the intrinsics of the circuit block and its operation
  - depends upon the excitation input

- **Verification**
  - whether functional or performance, relies on abstraction
  - for instance, extract a critical path of the circuit
  - results independent of the excitation input
    - but relies on a large number of assumptions (due to necessary abstraction) which may or may not hold
    - restrictive application

- **Electrical Verification**
  - approaches that can not quite adopt simulation
  - use rules, abstractions to ensure “good” design / “Circuit debuggers”
Terminology

- **Timing Verification**
  - simplified models over circuit simulation models, e.g., PathMill

- **Functional, Formal Verification**
  - build a model of the implemented circuits behavior
  - compare this behavior against specification
    - or another implementation
  - check this behavior for absence/presence of some properties
    - generally “safety” properties
  - prove that the circuit works as specified (=intended)
Formal Verification

- **Equivalence checking**
  - Combinational circuits
  - Sequential circuits

- **Model checking**
  - Problem statement
  - Explicit algorithms (on graphs)
  - Symbolic algorithms (using BDDs)

- **Theorem proving**
  - Deductive reasoning
Design Verification

Done at multiple levels and against multiple implementations

- **behavior**
  - Design 1: HDL / RTL ≡ ?
  - Design 2: RTL

- **function**
  - Design 1: Logic level ≡ ?
  - Design 2: Logic level

- **structure**
  - Design 1: Gate level ≡ ?
  - Design 2: Gate level

- **layout**
  - Design 1: Mask level ≡ ?
  - Design 2: Mask level
Why Formal Verification?

- Need for reliable hardware validation
- Simulation, test cannot handle all possible cases
- Formal verification conducts exhaustive exploration of all possible behaviors
  - compare to simulation, which explores some of possible behaviors
  - if correct, all behaviors are verified
  - if incorrect, a counter-example (proof) is presented
- Examples of successful use of formal verification
  - SMV system [McMillan 1993]
  - verification of cache coherence protocol in IEEE Futurebus+ standard
Binary Decision Diagrams

- **Binary Decision Diagram (BDD)**
  - compact data structure for Boolean logic
  - can represent sets of objects (states) encoded as Boolean functions
  - reduced ordered BDDs (ROBDD) are canonical
  - canonicity - essential for verification

- **Construction of ROBDD**
  - remove duplicate terminals
  - remove duplicate nodes (isomorphic subgraphs)
  - remove internal nodes with identical children
Construction of a Reduced Ordered BDD

\[ f = ac + bc \]

Truth table:

<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>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:

- **f**
- **a**
- **b**
- **c**

- 1 edge
- 0 edge

The truth table shows the values of \( f = ac + bc \) for different combinations of \( a, b, \) and \( c \). The decision tree represents the logic function \( f \) with nodes for \( a, b, \) and \( c \) and edges that reflect the logic operations of the function.
BDD Construction – cont’d

1. Remove duplicate terminals

2. Remove duplicate nodes

3. Remove redundant nodes

\[ f = (a+b)c \]
Use of ROBDDs in Formal Verification

- Equivalence of combinational circuits
- Canonicity property of BDDs:
  - if $F$ and $G$ are equivalent, their BDDs are identical (for the same ordering of variables)

$$F = a'bc + abc + ab'c$$

$$G = ac + bc$$

\[\equiv\]
Functional test generation

- Boolean satisfiability analysis (SAT)
  - to test for $H = 1$ (0), find a path in the BDD to terminal $1$ (0)
  - the path, expressed in function variables, gives a satisfying solution (test vector)

- More on it later.
Logic Manipulation using BDDs

- **Useful operators**
  - *Complement* \( \neg F = F' \)
    (switch the terminal nodes)
  - *Restrict*: \( F|_{x=b} = F(x=b) \)
    where \( b = \text{const} \)
Useful BDD Operators - cont’d

- **Apply**: \( F \odot G \)
  - where \( \odot \) stands for any Boolean operator (AND, OR, XOR, \( \rightarrow \))

Any logic operation can be expressed using only *Restrict* and *Apply*

Efficient algorithms that work directly on BDDs
Finite State Machines (FSM)

- FSM $M(X, S, \delta, \lambda, O)$
  - Inputs: $X$
  - Outputs: $O$
  - States: $S$
  - Next state function, $\delta(s, x) : S \times X \to S$
  - Output function, $\lambda(s, x) : S \times X \to O$
FSM Traversal

- **State Transition Graphs**
  - directed graphs with labeled nodes and arcs (transitions)
  - **symbolic** state traversal methods
    - important for symbolic verification, state reachability analysis, FSM traversal, etc.

- There are two important computations in FSMs
  - Image Computation: *Next* states from a given state
  - Pre-Image Computation: *Previous* states from a given set of states

- Both of these can be done efficiently using ROBDDs
Equivalence Checking

- Two circuits are *functionally* equivalent if they exhibit the same behavior.
- **Combinational circuits**
  - for all possible input *values*.
- **Sequential circuits**
  - for all possible input *sequences*. 
Combinational Equivalence Checking

- **Functional Approach**
  - transform output functions of combinational circuits into a unique (canonical) representation
  - two circuits are equivalent if their representations are identical
  - efficient canonical representation: BDD

- **Structural Approach**
  - identify structurally similar internal points
  - prove internal points (cut-points) equivalent
  - find implications
If BDD can be constructed for each circuit
- represent each circuit as *shared* (multi-output) BDD
  - use the same variable ordering!
- BDDs of both circuits must be *identical*

**Problem**
- Often BDDs are too large, not enough memory

**Remedy**
- Partition BDDs, check equivalences at internal points
- Dynamically order variables that lead to small size
- More sophisticated (data-oriented) decision diagrams
Sequential Equivalence Checking

- Represent each sequential circuit as an FSM
  - verify if two FSMs are equivalent
- **Approach 1**: reduction to *combinational* circuit
  - unroll FSM over \( n \) time frames (flatten the design)

Combinational logic: \( F(x(1,2, \ldots n), s(1,2, \ldots n)) \)

- check equivalence of the resulting combinational circuits
- problem: the resulting circuit can be too large to handle
Sequential Verification

- **Approach 2**: based on isomorphism of state transition graphs
  - two machines $M_1$, $M_2$ are *equivalent* if their state transition graphs (STGs) are *isomorphic*
  - perform state minimization of each machine
  - check if $\text{STG}(M_1)$ and $\text{STG}(M_2)$ are isomorphic
Approach 3: symbolic FSM traversal of the product machine

Given two FSMs: $M_1(X,S_1, \delta_1, \lambda_1, O_1)$, $M_2(X,S_2, \delta_2, \lambda_2, O_2)$

Create a product FSM: $M = M_1 \times M_2$
- traverse the states of $M$ and check its output for each transition
- the output $O(M) = 1$, if outputs $O_1 = O_2$
- if all outputs of $M$ are 1, $M_1$ and $M_2$ are equivalent
- otherwise, an error state is reached
- error trace is produced to show: $M_1 \neq M_2$
Define the product machine $M(X,S, \delta, \lambda, O)$

- states, $S = S_1 \times S_2$
- next state function, $\delta(s,x) : (S_1 \times S_2) \times X \to (S_1 \times S_2)$
- output function, $\lambda(s,x) : (S_1 \times S_2) \times X \to \{0,1\}$

- Error trace (*distiguishing sequence*)
  - that leads to an error state
  - sequence of inputs which produces 1 at the output of $M$
  - produces a state in $M$ for which $M_1$ and $M_2$ give different outputs

\[
\lambda(s,x) = \lambda_1(s_1,x) \oplus \lambda_2(s_2,x)
\]

\[
O = \begin{cases} 
1 & \text{if } O_1 = O_2 \\
0 & \text{otherwise}
\end{cases}
\]
Construction of the Product FSM

For each pair of states, \( s_1 \in M_1, s_2 \in M_2 \)
- create a combined state \( s = (s_1, s_2) \) of \( M \)
- create transitions out of this state to other states of \( M \)
- label the transitions (input/output) accordingly

Output = \{ 1 \) OK, 0 \) error \}
FSM Traversal in Action

**M1**

Initial states: $s_1=0$, $s_2=0$, $s=(0.0)$

<table>
<thead>
<tr>
<th>State reached</th>
<th>Out(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x=0$</td>
<td>$x=1$</td>
</tr>
</tbody>
</table>

- $New^0 = (0.0)$
- $New^1 = (1.1)$
- $New^2 = (0.2)$
- $New^3 = (1.0)$

- STOP - backtrack to initial state to get error trace: $x\{1,1,1,0\}$

**M2**

**Error state**
Model Checking

- Algorithmic method of verifying correctness
  - against temporal logic specifications
    - Formulae that contain time operators
  - A practical approach to formal verification

- Basic idea
  - System is described in a formal model
    - derived from high level design (HDL, C), circuit structure, etc.
  - The desired behavior is expressed as a set of properties
    - expressed as temporal logic specification
  - The specification is checked against the model
Temporal Logic

- Formalism describing sequences of transitions
- Time is not mentioned explicitly
- The temporal operators used to express temporal properties
  - eventually
  - never
  - always
- Temporal logic formulas are evaluated w.r.t. a state in the model
- Temporal operators can be combined with Boolean expressions
  - formulae can be written over state or over a path
Example of Temporal Formulae

- $E F (\ start \land \neg ready)$
  - eventually a state is reached where $start$ holds and $ready$ does not hold

- $A G (req \rightarrow A F ack)$
  - any time $request$ occurs, it will be eventually $acknowledged$

- $A G (E F restart)$
  - from any state it is possible to get to the $restart$ state
Model Checking Example

Traffic light controller (simplified)

C = car sensor
T = timer

C' + T'

Road 1

sensor

C Timer

Road 2

R1 Y2

G1 R2

R1 G2

G1 Y1

Y2

C T'

C T
Traffic light controller - Model Checking

- Model Checking task: check
  - safety condition
  - fairness conditions

- Safety condition:
  - no green lights on both roads at the same time

- Fairness condition:
  - eventually one road has green light

- To be continued…
Summary: Important Concepts

- What are basic components of verification?
  - Simulation, emulation, formal verification
  - Functionality versus performance verification

- When are these applied in a VLSI design project?

- Components of formal verification
  - Equivalence checking, model checking, theorem proving

- Modeling for equivalence checking
  - BDDs and ROBDDs: how do you construct these?
  - Using ROBDDs for Apply, Restriction

- Verification of sequential circuits: three approaches
  - Symbolic FSM traversal

- Model Checking: what is it?
  - Types of properties that are checked, examples.