The Verification Gap

- Verification determines whether a design satisfies its requirements (a.k.a. its specification):
  - Does it satisfy its functional requirements?
  - Does it satisfy its speed requirements?
  - etc.
- There is a growing gap between
  - the amount of verification that is desired, and
  - the amount that can be done
- The gap is caused by
  - Inadequate coverage with simulation
  - Approximate models (wire delays, for example)
  - etc.

Formal Verification Reduces the Gap

- Formal verification can give complete coverage
  - Mathematical techniques used to analyze all possible simulation vectors, without simulating them one by one
  - No test cases needed
- But formal verification cannot replace simulation
  - Current technology only effective for certain verification subtasks
- Using formal verification effectively requires understanding its strengths and weaknesses

Formal Verification vs Informal Verification

<table>
<thead>
<tr>
<th>Formal Verification</th>
<th>Informal Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete coverage</td>
<td>Incomplete coverage</td>
</tr>
<tr>
<td>Effectively exhaustive simulation</td>
<td>Limited amount of simulation</td>
</tr>
<tr>
<td>Cover all possible sequences of inputs</td>
<td>Spot check a limited number of input seq's</td>
</tr>
<tr>
<td>Check all corner cases</td>
<td>Some (many) corner cases not checked</td>
</tr>
<tr>
<td>No test vectors are needed</td>
<td>Designer provides test vectors (with help from tools)</td>
</tr>
</tbody>
</table>

Complete Coverage Example

- For these two circuits:
  \[
  f = ab(c+d) \\
  = abc + abd \\
  = g
  \]
- So the circuits are equivalent for all inputs
- Such a proof can be found automatically
- No simulation needed

Using Formal Verification

- No test vectors
- Equivalent to exhaustive simulation over all possible sequences of vectors (complete coverage)

Types of Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal</td>
<td>Requirements design should satisfy</td>
</tr>
<tr>
<td>Formal</td>
<td>Requirements are precise: a must for formal verification</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equivalence</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is one design equivalent to another?</td>
<td></td>
</tr>
<tr>
<td>Design has certain good properties?</td>
<td></td>
</tr>
</tbody>
</table>
Formal vs Informal Specifications

- **Formal requirement**
  - No ambiguity
  - Mathematically precise
  - Might be executable
- A specification can have both formal and informal requirements
  - Processor multiplies integers correctly (formal)
  - Lossy image compression does not look too bad (informal)

Types of Formal Verification

- **Property Checking**
- **Equivalence Checking**
  - Formal Verification
  - Different comb. equiv. methods give different market opportunities; must be understood for FV strategy

Equiv. Checking vs Property Checking

- **Equivalence checking**
  - Is one design equivalent to another?
  - One of the designs (the specification) is trusted
  - In practice, most useful at low levels of abstraction
- **Property checking**
  - Does the design have a given desirable property?
  - Properties are relatively small and easy to state, e.g.
    - Each packet sent is eventually acknowledged
    - Never more than one bus master
  - Do not need complete set of properties
  - Set of properties can evolve during design process
  - Most useful at high levels of abstraction
  - Finds bugs early

Types of Equivalence Checking

- Behavioral desc.
- RTL netlist
- Gate level netlist
- Trans. netlist
- Layout

Degree of Similarity: State Encoding

- Two designs have the same state encoding if
  - Same number of registers
  - Corresponding registers always hold the equal values
  - Register correspondence a.k.a. register mapping
  - Designs have the same state encoding if and only if
    - there exists a register mapping
  - Greatly simplifies verification
    - If same state encoding,
      - then combinational equivalence algorithms can be used

Producing the Register Mapping

- By hand
  - Time consuming
  - Error prone
  - Can cause misleading verification results
- Side-effect of methodology
  - Mapping maintained as part of design database
- Automatically produced by the verification tool
  - Minimizes manual effort
  - Depends on heuristics
Degree of Similarity: Combinational Nets

- Corresponding nets within a combinational block
- Corresponding nets compute equivalent functions
- With more corresponding nets
- Similar circuit structure
- Easier combinational verification
- Strong similarity
  - If each and every net has a corresponding net in the other circuit,
  - then structural matching algorithms can be used

Degree of Similarity: Summary

Weak Similarity
- Different state encodings
- General sequential equivalence problem
- Expert user, or only works for small designs
- Same state encoding, but combinational blocks have different structure
- IBM’s Bool’sEye
- Compass’ VFormal

Strong Similarity
- Nearly identical structure: structural matching
- Compare gate level netlists (PBS, Chrysalis)
- Checking layout vs schematic (LVS)

Capacity of a Comb. Equiv. Checker

- Matching pairs of fanin cones can be verified separately
  - How often a gate is processed is equal to the number of registers it affects
  - Unlike synthesis, natural subproblems arise without manual partitioning
  - “Does it handle the same size blocks as synthesis?” is the wrong question
  - “Is it robust for my pairs of fanin cones?” is a better question
- Structural matching is easier
  - Blocks split further (automatically)
  - Each gate processed just once

User Needs

- Gate vs Gate (structural)
  - Post-synthesis step: verify netlist updates (scan insertion, buffers, etc)
- ASIC designers, ASSPs, ASIC vendor, Design Factories
- Limited debugging support required
- Gate vs Gate (nonstructural)
  - Manual optimization following synthesis
  - High performance, high volume design (microProc, fabless, ASSPs)
  - Requires good debugging support
  - No current robust commercial offering

User Needs (cont.)

- RTL vs Gate
  - Simulation mostly at RTL level, reduce simulation at gate level
  - ASIC designers, ASSPs, Design Factories and future DSM signoff
  - Sophisticated debugging support required (source level)
  - Methodology support required
    - Hierarchy and black boxes required for IP
    - Synthesis/simulation mismatch due to Synopsys don’t cares
  - Capacity and robustness are critical
  - No dominant player yet

User Needs (cont.)

- RTL vs RTL
  - IP resurrection and multiple revisions
  - Similar to RTL vs gate
  - Gate or RTL vs Transistor
  - Processor companies, library development
  - Key issue is robustness of automatic transistor extraction
Techniques

- Random simulation
  - Finds many unequal nets (but not all)
- OBDDs
  - Construct OBDDs representing all or part of a combinational block
  - Canonical form: cheap to compare
  - Potentially expensive to build
- Structural matching
  - Specialized techniques to quickly verify identical structure
- Decomposition points
  - Find matching internal nets, if they exist

Techniques (cont.)

- Pattern matching
  - Transform circuits to increase similarity
  - Examples: remove inverter pairs and buffers, use de Morgan’s laws
- Case splitting
  - Exhaustively consider all combinations of inputs to a block
  - A given case may leave some inputs undetermined
  - Therefore, many fewer than $2^n$ cases may be sufficient

Equivalence Checking: Research

- Early academic research into tautology checking
  - A formula is a tautology if it is always true
  - Equivalence checking: $f$ equals $g$ when $(f = g)$ is a tautology
  - Used case splitting
  - Ignored structural similarity often found in real world
- OBDDs [Bryant 1986]
  - Big improvement for tautology checking [Malik et. al 1988, Fujita et. al 1988, Coudert and Madre et. al 1989]
  - Still did not use structural similarity
- Using structural similarity
  - Combine with ATPG methods [Brand 1993, Kunz 1993]
  - Continuing research on combining OBDDs with use of structural similarity

Equivalence Checking: Tools

- Internal tools from processor companies
  - IBM (sold as BoolEye), Motorola, DEC, Intel, BULL, etc.
- VFormal from Compass
  - OBDD-based, licensed from BULL
- CheckOff-E from Abstract Hardware
  - OBDD-based sequential equivalence checker
- Design VERIFYer from Chrysalis
  - No OBDDs, but “symbolic logic” is only a slight extension of the netlist data structures used in synthesis