Advanced Verification Techniques for DO-254
Mike Bartley (mike@testandverification.com)

Test and Verification Solutions
Delivering Tailored Solutions for Hardware Verification and Software Testing
Agenda

- Some background

- Advanced Verification Techniques

- Compliance
  - Hardware
  - (Software)

- Combining Advanced Verification Techniques with Compliance
A Quick Intro to your Speaker

- PhD in Mathematical Logic

- Worked in software testing and hardware verification for over 25 years
  - IPL, Praxis, ST-Micro, Infineon, start-ups (Elixent and ClearSpeed)
  - TVS

- Started TVS in 2008
  - Software testing and hardware verification products and services
TVS – Leaders in Testing and Verification

- Deliver services closer to our customers
- Services where costs & staff availability are important factors
- Run projects on client sites or off-site
- Help customers implement off-shore verification and testing
## Customers and Customer Retention

<table>
<thead>
<tr>
<th>Company</th>
<th>Retention Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcom</td>
<td>2 years</td>
</tr>
<tr>
<td>Infineon</td>
<td>5 years</td>
</tr>
<tr>
<td>Intel</td>
<td>2 years</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>4 years</td>
</tr>
<tr>
<td>NXP</td>
<td>2.5 years</td>
</tr>
<tr>
<td>ST</td>
<td>3 years</td>
</tr>
</tbody>
</table>
Some “advanced” verification techniques

- Constrained Random
- Functional Coverage
- Code Coverage
- Formal Verification
- Regression results metrics
- Bug rate analysis
- Analysis of open issues
- Code review completion
- Mutation analysis
- Software running
- Independent verification team
- Are all requirements verified?

“Verification - it's all about confidence”
Mike Bartley, SNUG 2001

Which ones to adopt?
The Human Factor in Verification
Why do we need separate verification team?

- Errors are introduced by *(mis)*interpretation.

Assumes you have a specification!

DANGER: When a designer verifies her/his own design – then she/he is verifying her/his own interpretation of the design.
Functional Verification Trends

*Industry evolving its functional verification techniques*

- **Assertions**: 2007 - 37%, 2010 - 69%
- **Constrained-Random Simulation**: 2007 - 41%, 2010 - 64%
- **Code coverage**: 2007 - 48%, 2010 - 72%
- **Functional coverage**: 2007 - 40%, 2010 - 72%

The adoption of formal property checking has grown by 53%
The mechanics of an advanced test bench

Stimulus generator

Constraint
addr data

Test

Driver

Design Under Test

assert

Coverage

Monitor

Functional Coverage

Checker

Assertions

Active Passive

Code Coverage
What is functional Coverage? Examples

- The system may transfer packets of different sizes
  - The test plan may require that transfer sizes with the following size or range of sizes be observed:
    - 1, 2, 3, 4 to 127, 128 to 252, 253, 254, or 255

- Functional coverage also examines the relationships between different objects
  - Cross coverage
  - An example of this would be examining whether an ALU has done all of its supported operations with every different input pair of registers
  - And if the ALU has written back to an input register
Adding value to your current test bench
Add advanced techniques to your current test bench

<table>
<thead>
<tr>
<th>Technique</th>
<th>Effort</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Coverage</td>
<td>• Low effort to start measuring&lt;br&gt;• High effort to “sign-off” holes</td>
<td>• Very useful when &lt; 100%&lt;br&gt;• When 100% - need other data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional Coverage</td>
<td>• High effort to define a full coverage model&lt;br&gt;• High effort to implement the coverage model&lt;br&gt;• High effort to “sign-off” holes</td>
<td>• Check that major features are fully verified</td>
</tr>
<tr>
<td>Assertions</td>
<td>• Effort varies with number of assertions</td>
<td>• High value with well defined assertions&lt;br&gt;• High value for debug</td>
</tr>
<tr>
<td>Checker</td>
<td>• Effort varies with sophistication of the checker</td>
<td>• High value – can write tests more quickly. Can consider pseudo random</td>
</tr>
<tr>
<td>Constrained random</td>
<td>• High effort – complex&lt;br&gt;• Needs a checker and final coverage</td>
<td>• Very high</td>
</tr>
</tbody>
</table>
The mechanics of finding a bug in simulation

Stimulate

…..01010101
…..01001101
…..10011010
…..01100101
…..11110101
…..00010101

Design Under Test

Propagate

…..01010101
…..01001101
…..10011010
…..01100101
…..11110101
…..00010101

Observe

Mutation testing adds value in terms of test suite qualification.

Actual Results

Expected Results

Compare
## Add advanced techniques to your current test bench

<table>
<thead>
<tr>
<th>Technique</th>
<th>Effort</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Coverage</td>
<td>• Low effort to start measuring</td>
<td>• Very useful when &lt; 100%</td>
</tr>
<tr>
<td></td>
<td>• High effort to “sign-off” holes</td>
<td>• When 100% - need other data</td>
</tr>
<tr>
<td></td>
<td>• High effort to implement the coverage model</td>
<td>• Check that major features are fully verified</td>
</tr>
<tr>
<td></td>
<td>• High effort to “sign-off” holes</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>• High effort to define a full coverage model</td>
<td>• High value with well defined assertions</td>
</tr>
<tr>
<td>Coverage</td>
<td>• High effort to implement the coverage model</td>
<td>• High value for debug</td>
</tr>
<tr>
<td></td>
<td>• High effort to “sign-off” holes</td>
<td></td>
</tr>
<tr>
<td>Assertions</td>
<td>• Effort varies with number of assertions</td>
<td>• High value - High value – can write tests more quickly. Can consider pseudo random</td>
</tr>
<tr>
<td>Checker</td>
<td>• Effort varies with sophistication of the checker</td>
<td></td>
</tr>
<tr>
<td>Constrained</td>
<td>• High effort – complex</td>
<td>• Very high</td>
</tr>
<tr>
<td>random</td>
<td>• Needs a checker and fnal coverage</td>
<td></td>
</tr>
<tr>
<td>Mutation Analysis</td>
<td>• Low effort to adopt a tool</td>
<td>• Very high if using tool – discover quality of you verif.</td>
</tr>
<tr>
<td></td>
<td>• High effort to run and analyse output</td>
<td>• “DIY” will give useful feedback</td>
</tr>
<tr>
<td></td>
<td>• Low effort for “Do It Yourself”</td>
<td></td>
</tr>
</tbody>
</table>
The rise of design IP

External IP increase by 138% from 2007 to 2010
Wilson Research Group and Mentor Graphics
2010 Functional Verification Study
Functional Verification Approaches

Verification

Static
- Reviews
- Code Analysis
  - Linters
- Equivalence Checking

Dynamic
- Formal
  - Dynamic Formal
- Simulation
- Prototyping
  - Silicon
  - FPGA
- Emulation
- Theorem Proving
- Model Checking
Formal Verification: Some example properties

- The adoption of formal property checking has grown by 53%

- `a_busy` and `b_busy` are never both asserted on the same cycle
- If the input ready is asserted on any cycle, then the output start must be asserted within 3 cycles
- If an element with tag `t` and data value `d` enters the block, then the next time that an element with tag `t` leaves the block, its data value is the same as the output of a reference piece of combinatorial logic for which the input is `d`
- `stall` cannot remain high indefinitely

Can be checked during simulation (but not proved by simulation)

A liveness property
# Model Checking – a brief introduction

## Inputs to the tool

<table>
<thead>
<tr>
<th>3 inputs to the tool</th>
<th>For example</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A model of the design</td>
<td>– Usually RTL</td>
</tr>
<tr>
<td>• A property or set of properties representing the requirements</td>
<td>– Items are transmitted to one of three destinations within 2 cycles of being accepted</td>
</tr>
<tr>
<td>• A set of assumptions, expressed in the same language as the properties</td>
<td>– The request signal is stable until it is granted</td>
</tr>
<tr>
<td>• typically constraints on the inputs to the design</td>
<td>• (req_in &amp; &amp; gnt_in)</td>
</tr>
<tr>
<td></td>
<td>(rec_a</td>
</tr>
<tr>
<td></td>
<td>• (req_in &amp; &amp; !gnt_out)</td>
</tr>
<tr>
<td></td>
<td>• We would of course need a complete set of constraints</td>
</tr>
</tbody>
</table>
Model Checking – a brief introduction

Outputs from the tool

- **Proved**
  - the property holds for all valid sequences of inputs

- **Failed\((n)\)**
  - there is at least one valid sequence of inputs of length \(n\) cycles, as defined by the design clock, for which the property does not hold.
  - In this case, the tool gives a waveform demonstrating the failure.
  - Most algorithms ensure that \(n\) is as small as possible, but some more advanced algorithms don’t.

- **Explored\((n)\)**
  - there is no way to make the property fail with an input sequence of \(n\) cycles or less
  - For large designs, the algorithm can be expensive in both time and memory and may not terminate
The Strengths of Model Checking

- **Ease of set-up**
  - No test bench required, add constraints as you go, VIP?

- **Flexibility of verification environment**
  - Constraints can be easily added or removed

- **Full proof**
  - Of the properties under the given constraints
  - (Can also prove “completeness” of the properties)

- **Intensive stressing of design**
  - Explored\(n\) constitutes a large amount of exploration of the design
  - Judgement when the number of cycles explored in a run is sufficient
  - *Significant bugs already found within a this number of cycles*

- **Corner cases**
  - Find any way in which a property can fail (under the constraints)
Potential issues with formal verification

- **False failures**
  - Need constraints to avoid invalid behaviour of inputs

- **False proofs**
  - Bugs may be missed in an over-constrained environment.

- **Limits on size of the model that can be analysed**

- **Non-exhaustive checks: \textit{Explored}(n)**
  - Interpret the results
  - \textit{Can require significant knowledge and skill}

- **Non-uniform run times**
  - Often it cannot be predicted how long it will take for a check either to terminate or to reach a useful stage

\textbf{This can make formal unpredictable!}
Safety-critical Systems

“\textbf{A safety critical system is a system where human safety is dependent upon the correct operation of the system}”

\textbf{Elements of safety critical systems:}
  - Computer hardware
  - Other electronic and electrical hardware
  - Mechanical hardware
  - Operators or users
  - Software

\textbf{Traditionally associated with embedded control systems}
Safety Standards

- IEC61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
- IEC60880: Software aspects for computer-based systems performing category A functions
- DO178: Software considerations in airborne systems and equipment certification
- DO254: Design Assurance Guidelines for Airborne Electronic Hardware
- EN50128: Software for railway control and protection systems
- IEC62304: Medical device software -- Software life cycle processes
- ISO26262: Road vehicles – Functional safety
Safety Standards

Process objectives and outputs
Integrity levels/classes

Picture from Kyle Beane http://www.noisefestival.com/node/14294
DO-254 identifies the following data items

- Hardware Verification Plan
- Validation and Verification Standards
- Hardware Traceability Data
- Hardware Review and Analysis Procedures
- Hardware Review and Analysis Results
- Hardware Test Procedures
- Hardware Test Results
- Hardware Acceptance Test Criteria
- Problem Reports
- Hardware Configuration Management Records
- Hardware Process Assurance Records
DO-254 identifies the following data items

- Hardware Verification Plan
- Validation and Verification Standards
- Hardware Traceability Data
- Hardware Review and Analysis Procedures
- Hardware Review and Analysis Results
- Hardware Test Procedures
- Hardware Test Results
- Hardware Acceptance Test Criteria
- Problem Reports
- Hardware Configuration Management Records
- Hardware Process Assurance Records
A closer look

- **Hardware Verification Plan**
  - The hardware verification plan describes the procedures, methods and standards to be applied and the processes and activities to be conducted for the verification of the hardware items.

- **Hardware Traceability Data**
  - Hardware traceability establishes a correlation between the requirements, detailed design, implementation and verification data to support configuration control, modification and verification of the hardware item.
ISO 26262 Requirements Management

ISO 26262 Stipulates
“The management of safety requirements includes managing requirements, obtaining agreement on the requirements, obtaining commitments from those implementing the requirements, and maintaining traceability.”

Diagram:
- Requirements
- Intent to implement
- Intent to verify
- Proof of implementation
- Stakeholder Requirements (Customers and internal)
- Product Requirements
- Safety Requirements
- System and Module Specs
- Verification & Test Plans
- Verification & Test Results
REFINING THE REQUIREMENTS TO TEST DESCRIPTION LEVEL

Feature Level Requirements
(Top-Level test Plan)

Req1

Refined requirements
(sub-features)

Req1.1

Req1.2

Refined requirements
(sub-features and goals)

Req1.1.1

Req1.1.2

Req1.2.1

Goal1.2.2

Measurable goals

Goal1.1.1.1

Goal1.1.1.2

Goal1.1.2.1

Goal1.2.1.1
COMPLIANCE : HIERARCHICAL SET OF REQUIREMENTS
What are the implications for Requirements Signoff?

- **Just mapping a requirement to a directed test is NOT sufficient**
- **Requirements need to map to**
  - Tests
    - Directed
    - Constrained random with a particular seed
  - Coverage
    - Code, functional and assertion
  - Checkers
    - Dynamic and Static
  - Proofs

- **Need to automate**
  - Test pass and fail
  - Coverage collection and reporting
  - Checker pass and fail

- **All linked to configuration management data**
Using Data From Advanced Verification

EDA → DB → UCIS API

Doors

Requirements → Test Plan

This is VERY hard (DXL?)

Can be done. But hard to update with results
Using Data From Advanced Verification

EDA → DB → UCIS API

Doors → asureSIGN

Requirements → Test Plan → DB

Can be done easily in asureSIGN

This is Done
# Mapping Goals to Coverage or Tests

![Screenshot of software interface showing mapping of goals to tests]
• Is a key indicator of the overall progress of the project.

• Any dip or peaks indicate debugging and corrective actions may be required.
Summary

• Increasing design complexity requires more advanced verification techniques

• DO254 will allow for such techniques BUT

• We need to ensure we can still map requirements to verification results
  – This is a lot more complex than for directed testing
Contact details

- Mike Bartley
- mike@testandverfication.com
- Mobile: +44 (0) 7796 307958
- Fax: 0117 903 9001