Trends in Functional Verification
Your speaker: Mike Bartley

- PhD in Mathematical Logic
- MSc in Software Engineering
- MBA

- Worked in software testing and hardware verification for over 25 years
  - ST-Micro, Infineon, Panasonic, ARM, NXP, nVidia, ClearSpeed, Gnodal, DisplayLink, Dialog, ...
  - Worked in formal verification of both software and hardware

- Started TVS in 2008
  - Software testing and hardware verification products and services
  - Offices in India, UK, France and Germany
Proposed agenda (subset of 1 day training)

- The challenges of verification
  - Taken from TVS “Verification Futures” conferences

- Advanced verification techniques

- Trends in functional verification

- Some info on TVS
<table>
<thead>
<tr>
<th>Speaker</th>
<th>Topic</th>
<th>Specific Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bryan Dickman, ARM</td>
<td>Complexity</td>
<td>Design for verification</td>
</tr>
<tr>
<td></td>
<td>Scalability</td>
<td>Meeting our need for cycles</td>
</tr>
<tr>
<td>Olivier Haller, ST</td>
<td>Scalability</td>
<td>Is exhaustive RTL simulation scalable?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data management</td>
</tr>
<tr>
<td>Hans Lunden, Ericsson</td>
<td>TLM in verification</td>
<td>Improved TTM</td>
</tr>
<tr>
<td></td>
<td>VIP</td>
<td>Improved quality, Make or buy?</td>
</tr>
<tr>
<td>Clemens Muller, Infineon</td>
<td>Complexity</td>
<td>Mastering the verif complexity</td>
</tr>
<tr>
<td></td>
<td>Debug Automation</td>
<td>Managing all the data</td>
</tr>
<tr>
<td></td>
<td>Requirements driven verification</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
<td>Scalability</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Bryan Dickman, ARM</td>
<td>- Design for verification</td>
<td>- Meeting our need for cycles</td>
</tr>
<tr>
<td>Olivier Haller, ST</td>
<td>- Scalability</td>
<td>- Complexity</td>
</tr>
<tr>
<td></td>
<td>- Is exhaustive RTL simulation scalable?</td>
<td>- Nobody understands the full system</td>
</tr>
<tr>
<td></td>
<td>- Data management</td>
<td></td>
</tr>
<tr>
<td>Hans Lunden, Ericsson</td>
<td>- TLM in verification</td>
<td>- VIP</td>
</tr>
<tr>
<td></td>
<td>- Improved TTM</td>
<td>- Improved quality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Make or buy?</td>
</tr>
<tr>
<td>Clemens Muller, Infineon</td>
<td>- Complexity</td>
<td>- Debug Automation</td>
</tr>
<tr>
<td></td>
<td>- Mastering the verification complexity</td>
<td>- Managing all the data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **Top1: Mastering Verification Complexity**
  
  – Continuous increase in number of IP’s and embedded processors
    * 2006: 30-40 IP’s, 1 CPU
    * 2011: 80+ IP’s, 6+ CPU’s
    * 2016: 120+ IP’s, 20 CPU’s ?
  
  – The more IP’s the higher the risk of late spec & implementation changes
  
  – Driving towards true Hw/Sw Co-Verification
  
  – Reuse of verification environments / stimulus from IP-level into big multi-CPU SoC environments
<table>
<thead>
<tr>
<th>Speaker</th>
<th>Complexity</th>
<th>Scalability</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bryan Dickman, ARM</td>
<td>Design for verification</td>
<td>Meeting our need for cycles</td>
<td>How do we know when we are done?</td>
</tr>
<tr>
<td>Olivier Haller, ST</td>
<td>Scalability</td>
<td>Complexity</td>
<td>Productivity</td>
</tr>
<tr>
<td></td>
<td>Is exhaustive RTL simulation scalable?</td>
<td>Nobody understands the full system</td>
<td>Do more with same budget</td>
</tr>
<tr>
<td></td>
<td>Data management</td>
<td></td>
<td>Faster with derivatives</td>
</tr>
<tr>
<td>Hans Lunden, Ericsson</td>
<td>TLM in verification</td>
<td>VIP</td>
<td>Design for verification</td>
</tr>
<tr>
<td></td>
<td>Improved TTM</td>
<td>Improved quality</td>
<td>Quality and TTM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Make or buy?</td>
<td></td>
</tr>
<tr>
<td>Clemens Muller, Infineon</td>
<td>Complexity</td>
<td>Debug Automation</td>
<td>Requirements driven verification</td>
</tr>
<tr>
<td></td>
<td>Mastering the verif complexity</td>
<td>Managing all the data</td>
<td></td>
</tr>
</tbody>
</table>
2. Scalability

- Constrained-random simulation has been proven as a good bug-hunting flow, but...
  - How much simulation will be enough for a 10 GHz CPU?
  - How many cycles to verify 2 weeks at target speed of 1GHz?
  - Answer: $0.6 \times 10^{15}$

<table>
<thead>
<tr>
<th>Simulation (KHz)</th>
<th>Emulation (1 MHz)</th>
<th>FPGA (10 MHz)</th>
<th>Si (1 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target cycles $10^{15}$</td>
<td>1,000,000 sim slots</td>
<td>1000 emulation slots</td>
<td>100 FPGA slots</td>
</tr>
<tr>
<td>Achievable cycles</td>
<td>$10^{11}$</td>
<td>$10^{12}$</td>
<td>$10^{14}$</td>
</tr>
</tbody>
</table>

- How will we scale simulation, emulation, FPGA to next gen of CPUs?
- What are the alternatives?
<table>
<thead>
<tr>
<th>Speaker</th>
<th>Topic</th>
<th>Subtopics</th>
<th>EDA Tool Maturity</th>
<th>Mixed Signal</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geoff Barrett, Broadcom</td>
<td>Scalability</td>
<td>At chip level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verification</td>
<td>Too much on diversions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andre Winkelmann, Wolfson</td>
<td>Requirements</td>
<td>Tracing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scalability</td>
<td>Ease of verifying derivatives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andrew Bond, NVidia</td>
<td>H/SW Co-</td>
<td>verification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>Verif</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Achieving Reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve Holloway, Dialog</td>
<td>Verification</td>
<td>Completion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
<td>Reducing verify complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tim Blackmore, Infineon</td>
<td>Complexity</td>
<td></td>
<td></td>
<td></td>
<td>Better use of sim cycles</td>
</tr>
<tr>
<td></td>
<td>Change</td>
<td>Making verify more agile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Topic</td>
<td>Details</td>
<td>EDA Tool Maturity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geoff Barrett, Broadcom</td>
<td>Scalability</td>
<td>• At chip level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verification resources</td>
<td>• Too much on diversions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andre Winkelmann, Wolfson</td>
<td>Requirements Tracing</td>
<td></td>
<td>Mixed Signal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scalloability</td>
<td>• Ease of verifying derivatives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andrew Bond, NVidia</td>
<td>H/SW Co-verification</td>
<td>• SW engineers avail to write real SW</td>
<td>Resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance Verif</td>
<td>• Everybody finds own solution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve Holloway, Dialog</td>
<td>Verification Completion</td>
<td>• Increasingly hard</td>
<td>Mixed Signal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Achieving Reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tim Blackmore, Infineon</td>
<td>Complexity</td>
<td>• Reducing verif complexity</td>
<td>Better use of sim cycles</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change</td>
<td>• Making verif more agile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recruitment

- Verification engineers are always in demand
- Even with some industry-wide unification of methodologies finding good engineers doesn’t seem to be getting easier
- With more design re-use and verification outsourcing flexible engineers seem harder to find
<table>
<thead>
<tr>
<th>Name</th>
<th>Issue/Achievement</th>
<th>EDA Tool Tool Maturity</th>
<th>Mixed Signal</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geoff Barrett, Broadcom</td>
<td>Scalability - At chip level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andre Winkelmann, Wolfson</td>
<td>Requirements Tracing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andrew Bond, NVidia</td>
<td>H/SW Co-verification - SW engineers avail</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve Holloway, Dialog</td>
<td>Verification Completion - Increasingly hard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tim Blackmore, Infineon</td>
<td>Complexity - Reducing verif complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Challenge 3 – Mixed Signal

- **MS verification made easy**
  - How do analogue and digital engineers work together?
  - Multitude of skills required

- **Boundary is fading**
  - Analogue verification incorporates digital techniques
  - Digital verification incorporates analogue features

- **Variety of modelling techniques and abstractions**

- **Power aware mixed signal verification**

- **UVM-AMS adoption**
<table>
<thead>
<tr>
<th>Name</th>
<th>Topic</th>
<th>Subtopics</th>
<th>Subtopics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent Arditi, ARM</td>
<td>Bug Avoidance</td>
<td>Functionally correct designs?</td>
<td>Improved hunting &amp; completion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proving absence of bugs</td>
</tr>
<tr>
<td>Thomas Goust, ST-E</td>
<td>Design Complexity</td>
<td>IC to chipset</td>
<td>Leading-Edge Tech</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple ICs</td>
<td>Outsourcing</td>
</tr>
<tr>
<td>Jerome Bombal, TI</td>
<td>HW-SW Co-Verification</td>
<td>Fast platform prototyping</td>
<td>Real-world functional coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christophe Chevallaz, ST</td>
<td>Verification Reuse</td>
<td>System Verification</td>
<td>Verification Mgt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lots of opportunity</td>
<td>Data mgt</td>
</tr>
<tr>
<td>France 2012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Laurent Arditi, ARM</strong></td>
<td><strong>Bug Avoidance</strong>&lt;br&gt;• Functionally correct designs?</td>
<td><strong>Bug Hunting</strong>&lt;br&gt;• Improved hunting &amp; completion</td>
<td><strong>Bug Absence</strong>&lt;br&gt;• Proving absence of bugs</td>
</tr>
<tr>
<td><strong>Thomas Goust, ST-E</strong></td>
<td><strong>Design Complexity</strong></td>
<td><strong>IC to chipset</strong>&lt;br&gt;• Multiple ICs</td>
<td><strong>Leading-Edge Tech</strong>&lt;br&gt;• Outsourcing</td>
</tr>
<tr>
<td><strong>Jerome Bombal, TI</strong></td>
<td><strong>HW-SW Co-Verification</strong></td>
<td><strong>Fast platform prototyping</strong></td>
<td><strong>Real-world functional coverage</strong></td>
</tr>
<tr>
<td><strong>Christophe Chevallaz, ST</strong></td>
<td><strong>Verification Reuse</strong>&lt;br&gt;• Lots of opportunity</td>
<td><strong>System Verification</strong></td>
<td><strong>Verification Mgt</strong>&lt;br&gt;• Data mgt</td>
</tr>
</tbody>
</table>
Challenge 3 – Verification Management

- The challenge to manage huge amount of verification data
  - Amount of verification data make more complex the risk decision of verification closure

- Some Directions partially or to be implemented
  - Refine the verification Metrics
  - Merge the metrics (SOC / IPS – various source)
  - Usage of MySQL data Base
  - Leverage on Business Intelligence tool to support Verification Closure
  - Define metrics on non-functional properties (performance, power, energy, temperature, ...)
<table>
<thead>
<tr>
<th>Name</th>
<th>Topic</th>
<th>Verification Environment</th>
<th>Debugging</th>
<th>3rd Party IP Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin Ruhwandl, Lantiq</td>
<td>Multi-Language Verif environments</td>
<td></td>
<td></td>
<td>More automation</td>
</tr>
<tr>
<td>Michael Rohleder, Freescale</td>
<td>Synthesis/Timing Constraints</td>
<td>Holistic Coverage</td>
<td></td>
<td>And VIP</td>
</tr>
<tr>
<td>Wolfgang Ecker, Infineon</td>
<td>Requirements driven verification</td>
<td>TopDown/BottomUp</td>
<td></td>
<td>Heterogeneous Systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Digital, Analog, FW</td>
</tr>
<tr>
<td>Germany 2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Martin Ruhwandl, Lantiq** | **Multi-Language Verif environments** | **Debugging**  
- More automation | **3rd Party IP integration**  
- And VIP |
| **Michael Rohleder, Freescale** | **Synthesis/Timing Constraints** | **Holistic Coverage**  
- Combining views | **Disconnected Views**  
- Functional, timing, power, SW |
| **Wolfgang Ecker, Infineon** | **Requirements driven verification** | **TopDown/BottomUp**  
- Verif at right level | **Heterogeneous Systems**  
- Digital, Analog, FW |
Wolfgang Ecker, Infineon

- **Required by ISO 26262**
  - “Road vehicles – Functional safety” and other similar standards

- **Validate the verification**
  - Have the right things been verified
  - Avoid that requirements haven’t been verified and things have been verified, that haven’t been required

- **Reuse implementation of verification goal**

- **Keep track with change requests**

- **Enable impact analysis**
<table>
<thead>
<tr>
<th>Name</th>
<th>AMS Verification</th>
<th>Dynamic Power Verif</th>
<th>Timing Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sainath Karlapalem, NXP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Udaya Kumar Napa, MaxLinear</td>
<td>Coverage Closure</td>
<td>Integrating Levels of Verif</td>
<td>Requirements driven verification</td>
</tr>
<tr>
<td>Desikan Srinivasan, ARM</td>
<td>Exploiting Formal</td>
<td>System level coherency</td>
<td>Verif Schedule Predictability</td>
</tr>
</tbody>
</table>
## Europe 2013

<table>
<thead>
<tr>
<th>Chris Brown (Verification Manager), Broadcom, UK</th>
<th>Improved EDA competition through standardisation</th>
<th>Integration Testing: Improving IP to SoC verification reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon Bewick (Director ASIC Development), Ericsson, UK</td>
<td>Measuring Testbench quality</td>
<td>Fewer bugs, earlier bugs</td>
</tr>
<tr>
<td>Andy Walton, Altera, UK</td>
<td>UVM @ system-level but not at block-level isn’t</td>
<td>But we can just run it on the hardware</td>
</tr>
<tr>
<td>Tim Joyce (Verification Manager), STMicro, UK</td>
<td>Predicting and finding verification resources</td>
<td>Functional Coverage at SoC level</td>
</tr>
</tbody>
</table>
## France and Germany 2013

<table>
<thead>
<tr>
<th>Name and Organization</th>
<th>Activity</th>
<th>Tools</th>
<th>Strategy/Contractors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jerome Bombal (Director SOC Verification), Samsung LSI, France</td>
<td>Sign-off integration verification plans</td>
<td>Verification SW libraries</td>
<td>Integration coverage toolboxes</td>
</tr>
<tr>
<td>Beatrice Brochier and Yassine Elkhourassani, STMicro</td>
<td>Verifying Interconnect at top level</td>
<td>Verifying IOMUXing in a complex SoC</td>
<td></td>
</tr>
<tr>
<td>Martin Ruhwandl, Infineon Technologies</td>
<td>ISO 26262</td>
<td>Multi-site strategy/short term contractors</td>
<td>Analog-mixed signal engineers vs. digital engineers</td>
</tr>
</tbody>
</table>
# Analysis

<table>
<thead>
<tr>
<th>Category</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>7</td>
</tr>
<tr>
<td>Integrating Languages, Views and Techniques</td>
<td>7</td>
</tr>
<tr>
<td>Completeness</td>
<td>5</td>
</tr>
<tr>
<td>Productivity</td>
<td>5</td>
</tr>
<tr>
<td>Requirements Driven Verif/ISO 26262</td>
<td>5</td>
</tr>
<tr>
<td>Scalability</td>
<td>4</td>
</tr>
<tr>
<td>Reuse</td>
<td>4</td>
</tr>
<tr>
<td>System</td>
<td>4</td>
</tr>
<tr>
<td>Mixed Signal</td>
<td>4</td>
</tr>
<tr>
<td>HW/SW</td>
<td>3</td>
</tr>
<tr>
<td>Resources</td>
<td>3</td>
</tr>
<tr>
<td>Integration Verification</td>
<td>3</td>
</tr>
<tr>
<td>Design for Verif</td>
<td>2</td>
</tr>
<tr>
<td>Debug</td>
<td>2</td>
</tr>
<tr>
<td>Demonstrating Bug Absence</td>
<td>2</td>
</tr>
<tr>
<td>Synthesis/Timing Constraints</td>
<td>2</td>
</tr>
<tr>
<td>Power Verification</td>
<td>2</td>
</tr>
<tr>
<td>FPGA Specific</td>
<td>2</td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
</tr>
<tr>
<td>Change</td>
<td>1</td>
</tr>
<tr>
<td>Leading Edge Technology</td>
<td>1</td>
</tr>
<tr>
<td>Verification Data Mgt</td>
<td>1</td>
</tr>
<tr>
<td>Predictability</td>
<td>1</td>
</tr>
<tr>
<td>EDA tool Integration</td>
<td>1</td>
</tr>
<tr>
<td>Measuring Test Bench Quality</td>
<td>1</td>
</tr>
<tr>
<td>IO Muxing at SoC Level</td>
<td>1</td>
</tr>
</tbody>
</table>
The Basic Problem

- Presented by Harry Foster
- At the TVS UK Conference “Verification Futures” 2011

Bell Statisticians (1910):
Due to the rapid adoption of the telephone, every woman in America would soon be required as switchboard operators

Source: SEMATECH 1997

- Silicon Density Doubles Every 18 Months
- Design Productivity Doubles Every 39 Months
- Verification Productivity

Bell Statisticians (1910):
Due to the rapid adoption of the telephone, every woman in America would soon be required as switchboard operators
## ARM Statistics Mike Muller keynote at DAC 2012

<table>
<thead>
<tr>
<th>Measure</th>
<th>ARM1 1983</th>
<th>Cortex-M0 2012</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>Similar transistor counts</td>
<td></td>
<td>So good candidate for comparison</td>
</tr>
<tr>
<td>Size</td>
<td>X</td>
<td>X/10,000</td>
<td>13 process generations apart – so this is expected</td>
</tr>
<tr>
<td>Speed</td>
<td>Y</td>
<td>Y*16</td>
<td>Should be 64 (6 performance scalings)</td>
</tr>
<tr>
<td>Power</td>
<td>5V</td>
<td>950mV</td>
<td>Should be 8mV This explains lower than expected speed improvements</td>
</tr>
<tr>
<td>Layout</td>
<td>6 months</td>
<td>28 minutes</td>
<td>ARM1 was custom Cortex-M0 was automated synthesis and P&amp;R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measure</th>
<th>ARM1 1983</th>
<th>Dual Core Cortex-A15 2012</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>25K</td>
<td>150M</td>
<td></td>
</tr>
<tr>
<td>Design Productivity</td>
<td>6 person years (PY)</td>
<td>150 PY</td>
<td>240x increase in design productivity</td>
</tr>
</tbody>
</table>
## ARM Statistics Mike Muller keynote at DAC 2012

<table>
<thead>
<tr>
<th>Measure</th>
<th>ARM1 1983</th>
<th>Mali-T604</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software (LOC)</td>
<td>150 (assembly)</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>Software Productivity</td>
<td>.25 PY</td>
<td>190 PY</td>
<td>7x increase in productivity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measure</th>
<th>ARM1 1983</th>
<th>Cortex-M0 2012</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>24K</td>
<td>32K</td>
<td></td>
</tr>
<tr>
<td>Verification Effort</td>
<td>6 PY</td>
<td>11 PY</td>
<td></td>
</tr>
<tr>
<td>Machine Resources</td>
<td>2000 machine hours</td>
<td>1,439,000 machine hours</td>
<td>Taking the speed of the machines into account this is 3,000,000 times less efficient.</td>
</tr>
</tbody>
</table>
Static vs. Dynamic

Verification

Static

- Reviews
- Code Analysis
- Linters
  - Equivalence Checking

Dynamic

- Simulation
- Prototyping
  - Silicon
  - FPGA
  - Emulation
- Formal
  - Dynamic Formal
  - Model Checking
  - Theorem Proving
The mechanics of an advanced test bench

- **Test**
- **Stimulus generator**
  - constraint
  - addr
  - data
- **Driver**
- **Functional Coverage**
- **Monitor**
- **Checker**
- **Assertions**
- **Coverage**
- **Design Under Test**
  - Test
  - assert
- **Active**
- **Passive**

**Code Coverage**
Constrained Random Simulation

- Constrained random verification is now common
- Measured by coverage
  - Functional
  - Code

<table>
<thead>
<tr>
<th>Functional Coverage</th>
<th>Code Coverage</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Low</td>
<td>There is verification work to do.</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>Multi-cycle scenarios, corner cases, cross-correlations still to be covered.</td>
</tr>
</tbody>
</table>
| High                | Low          | Verification plan and/or functional coverage metrics inadequate.  
                       |                                                       | Check for “dead” code. |
| High                | High         | Increased confidence in quality.                  |

- Directed still plays a part – especially at SoC level

- But is this enough?
The rise of the Verification Language

HiLo → Verilog → SuperLog

VHDL

Vera

SV

Janick Bergeron

C → e

Building test benches – the rise of the methodology

Janick Bergeron

But what is the UVM?

- **UVM = Universal Verification Methodology**
  - Class Reference Manual
  - an open-source SystemVerilog base class library implementation
  - a User Guide

**methodology** = A system of broad principles or rules from which specific methods or procedures may be derived to interpret or solve different problems within the scope of a particular discipline. Unlike an algorithm, a methodology is not a formula but a set of practices.
What does “successful” adoption of UVM look like?

- **No bugs?**
  - Is this realistic?
  - How do you know “no bugs” before Tape-Out?
  - What is the cost and TTM impact of this target?

- **Everybody expert in UVM?**
  - Why? What about domain experts?
  - Are all test bench tasks of equal complexity?

- **Improved test bench quality?**
  - A UVM TB can only be as good as your verification plan!

- **Improved verification re-use**
  - Vertical, Horizontal, Divisional, Corporate, Industry

- **Improved test bench consistency?**
  - Improved “staff” re-use

- **Legacy**
  - Incorporation of legacy verification
How easy is UVM?

- **There’s More Than One Way To Do It**
  - Last time I looked the SV LRM had about 580 pages
  - And the UVM class reference guide had over 400 pages

- **Brian Hunter, Cavium = 180 page guidelines**

- **Beware of “Specman” zealots**

- **Easy to**
  - Lose consistent “look & feel”
  - Write non – reusable code
  - Use the wrong level of abstraction
  - Be “unsuccessful”
Why do we need a methodology?

- Building constrained random test benches is
  - Very complex
  - Very time consuming

- We need a layered approach
  - This will allow various skills to be applied

- Open up the scope for reuse

- Allow a market in VIP

- Ability to adapt the behaviour of the test bench
  - Language wars!

- Consistency of approach

- Portable skills between projects & companies
Some experiences of UVM adoption (Dialog Semi)

- **UVM constrained-random approach resulted in:**
  - High rates of bug discovery
  - Easier tracking of real progress
  - Managed verification closure

- **UVM won’t initially reduce your verification effort**
  - Until reuse is leveraged

- **Legacy directed tests can still add value**
  - OVM checking in passive mode

- **Engineers were able to get running quickly**
  - Application-specific examples & knowledge sharing
The mechanics of finding a bug in simulation

Stimulate

…..01010101
…..01001101
…..10011010
…..01001101

Design Under Test

Propagate

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

Mutation testing adds value in terms of test suite qualification.

Observe

Actual Results

Expected Results

Compare
Mutation Coverage

- Mutation coverage is designed to detect simple (typing) mistakes in the code
  - Wrong operator
    - + instead of –
    - >= instead of >
  - Wrong variable
  - Offset in loop boundaries

- A mutation is considered covered if we found a test that can distinguish between the mutation and the original
  - Strong mutation: the difference is visible in the primary outputs
  - Weak mutation: the difference is visible inside the DUV

It cannot detect missing functionality
Assertion-Based Verification (ABV)

- An **assertion** is an if statement with an error condition that indicates that the condition in the if statement is false.

- **Assertions have been used in SW design for a long time.**
  - `assert()` function part of C `#include <assert.h>`
  - Used to detect **NULL** pointers, out-of-range data, ensure loop invariants, etc.

- **Revolution through Foster & Bening’s OVL for Verilog.**
  - Clever way of encoding re-usable assertion library in Verilog. 😊

- **Assertions have become very popular for Design Verification in recent years:**
  - **Assertion-Based Verification** (also Assertion-Based Design)

- **Assertion coverage:** Measures how often an assertion condition has been evaluated.

- Assertions can also be used in formal verification
Inputs to Formal

- **3 inputs to the tool**
  - A model of the design
  - A property or set of properties representing the requirements
  - A set of assumptions, expressed in the same language as the properties
  - *typically constraints on the inputs to the design*

- **For example**
  - Usually RTL
  - Items are transmitted to one of three destinations within 2 cycles of being accepted
    - \((\text{req\_in} \&\& \text{gnt\_in}) \rightarrow \#\#[1;2] \quad (\text{req\_a} || \text{req\_b} || \text{req\_c})\)
  - The request signal is stable until it is granted
    - \((\text{req\_in} \&\& \neg \text{gnt\_out}) \rightarrow \#\#1 \quad \text{req\_in}\)
  - We would of course need a complete set of constraints
Model Checking – Outputs from the tools

- **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
Some example properties

- 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
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: Explored(n)**
  - Interpret the results
  - 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

This can make formal unpredictable!
The AHAA Model

- **Bug Avoidance**
  - The objective is to avoid adding bugs to designs in the first place.

- **Bug Hunting**:
  - The objective here is to find bugs in an implemented design

- **Bug Absence**:
  - The objective is to prove the absence of bugs

- **Bug Analysis**:
  - The time to analyse a bug
  - *Reproduce a failure (be it a test, assertion, property, etc),*
  - *Understand the root cause of the failure*
  - *Find a fix and prove it works*
  - *Ensure there is something in the regression environment to stop the bug returning.*
  - *Intelligent engineers also perform additional analysis to understand if any other bugs exist (as bugs tend to cluster).*
Strategic Issues with Formal

- **What simulation do I replace?**
  - Short answer is none unless block is done completely formally
  - The metrics are too different

- **We don’t know if or when it will complete**
  - Formal can take a long time to give very poor results

- **A high level of skill might be required**
  - To write the correct properties and constraints
  - To drive the tools
  - And to drive into bug avoidance in the future

- **So why bother?**
  - You can “get it for free” on the back of assertion-based verification
  - There are requirements that cannot be verified through simulation
  - *Cache coherency, liveness, deadlock,...*
  - We need it to cope with the increasing complexity of verification
Top Level Test Generation for CPU

- **Bias tests to hit interesting corner cases**
  - Scenario interleaving
  - Target shared resources/’points of convergence’

- **Non-repetitive useful tests**

- **There should be an efficient workflow**
  - Generation performance
  - Target diverse platforms
  - Ease of use
  - Maintainability
  - Reuse (of testing knowledge)
  - Effective result checking:
    - *Propagation of results*
    - *Trace comparison*
Top Level Test Generation for BFM

- **Bias tests to hit interesting corner cases**
  - Scenario interleaving
  - Target shared resources/’points of convergence’

- **Non-repetitive useful tests**

- **There should be an efficient workflow**
  - Generation performance
  - Target diverse platforms
  - Ease of use
  - Maintainability
  - Reuse (of testing knowledge)
  - Effective result checking:
    - *Propagation of results*
    - *Trace comparison*
Main Concerns at SoC level

- **Synchronisation**
  - Between test running on CPU
  - And external VIP

- **Re-use from IP to SoC**
  - What do you want to re-use?
    - *Tests*
      - Why repeat IP level tests?
    - *Coverage models*
      - Why repeat IP level coverage?
    - *Drivers*
      - For traffic generation
    - *Protocol Checkers*
    - *Scoreboard*

- **Re-use from SoC to SoC**
  - How to isolate items that relate to the specific chip
  - *Such as memory map*
Some Specific SoC Verification Questions

- When I have one common test case suite used across multiple projects with slight modifications.
  - What is the best way of maintaining the tests?

- The significance of using “Coverage Driven” verification approach for SoC verification
  - Most of the times it is directed
  - Can we have a Constrained Random Environment?

- Any guidelines for selection of the Verification language for any environment
  - IP level
  - SoC Testbench
  - SystemC, SV, Specman?
Requirements Driven Signoff – ISO26262 Compliance

- Stakeholder Requirements
- (Customers and internal)
- Product Requirements
- Product Architecture
- Product Specification and Features
- Verification & Test Plans
- Verification & Test Results

- Intent to implement
- Intent to verify
- Proof of implementation
- Requirements
Metrics can be:
- From HW verification
- From Silicon validation
- From SW testing

Export Reqs Status as XML:
Req1 [×,√]  
Req2 [×,√]
Completion Criteria – Metrics to Measure Confidence

- Coverage Numbers
- Bug rate drop
- Resolution of open issues
- Reviews complete
- Regression results
- Mutation analysis
- Software running on FPGA
- No failing properties
- Are all requirements tested
  - Which ones can be dropped?

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

*Trends: Number of gates of logic and datapath, excluding memories*

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission
Verification Reuse Trends

Mean testbench composition trends

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission
Verification Languages

Trends in languages used for verification (testbenches)

SystemVerilog grew 271% between 2007 and 2012

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

Languages Used for Verification (testbenches)

- Multiple answers possible
Design and Verification Languages
Trends in testbench methodologies and base-class libraries

486% UVM growth between 2010 and 2012
46% UVM projected growth in the next twelve months
Half of the designs over 5M gates use UVM

Testbench Methodologies and Base-Class Libraries

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

* Multiple answers possible
Effort and Results

Percentage of Non-FPGA total project time spent in verification

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission
More and More Verification Engineers

Average Peak Number of Design vs. Verification Engineers

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

~ 1-to-1 ratio of peak design and verification engineers
Effort and Results

Mean time Non-FPGA verification engineers spends in different tasks

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

More time spent in debug than any other task!
Functional Verification Trends

Industry evolving its functional verification techniques

- Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

• Multiple answers possible
Summary

- Challenges
- A variety of verification methodologies
- It’s all about confidence
- The trends