Test and Verification Solutions

Resistance is Futile: Learning to love UVM!

Experts In Verification
The Verification Challenge
Effort Spent On Verification

Trend in the percentage of total project time spent in verification

Wilson Research Group and Mentor Graphics 2010 Functional Verification Study, Used with permission
Effort Spent On Verification

Mean peak number of designers vs. verification engineers

4% increase in designers vs. 58% increase in verification engineers

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study,
Used with permission
Harry Foster, Mentor Graphics
Verification Futures: The Next Five Years,
Nov 15, 2011

Moore’s Law

Murphey’s Law
Functional Verification Trends

*Industry evolving its functional verification techniques*

![Bar chart showing trends in functional verification techniques over time.](chart.png)

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

Wilson Research Group and Mentor Graphics 2010 Functional Verification Study, Used with permission

*The adoption of formal property checking has grown by 53%*
UVM to the rescue?
SystemVerilog adoption has increased by 233% in the past three years!
UVM is expected to grow by 286% in the next 12 months!

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission
UVM has a great pedigree
But what is the UVM?

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

**meth·od·ol·o·gy** = 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.
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

• Easy to
  – Lose consistent “look & feel”
  – Write non – reusable code
  – Use the wrong level of abstraction

• Brian Hunter, Cavium = 180 page guidelines
So why bother?

- The statistics show it is becoming the de-facto industry standard
  - Training is available
  - Engineers are available (market forces apply!)
  - Community of help
- Industry tools and VIP
- On-going maintenance (future proof)
- An open source community
- It CAN do the job
  - But roll out and adoption of UVM MUST be planned
Problems with Adoption (Dialog Semi)

- RTL-centric engineers learning OOP concepts
- Stimulus not constrained appropriately
- Checking at the wrong level of abstraction
  - Reference model in module-based “helper” code + assertions
- Dangerous use of configuration settings
  - set_config_int("*", "num_agents", …);
- Slippage between Vplan & coverage model
- Derivative projects could not reuse agents easily
  - Tightly coupled to interface
- Module – chip reuse is non-trivial

From VerificationFutures 2011
Reproduced with permission from Dialog Semi
Some solutions (Dialog Semi)

- Encapsulate VIP settings in configuration objects
- Encapsulation of BFM tasks in interface
  - Better reuse model for derivative DUTs with changing i/f
- Structure of Scoreboard for reuse & decoupled checks
  - E.g. MVC pattern
- Leverage common sequence API (e.g. register-based)
- Review process essential to ensure consistent verification approach
- Multi-layered approach to verification
  - Infrastructure & VIP development
  - Project specific stimulus, checks and coverage

From VerificationFutures 2011
Reproduced with permission from Dialog Semi
Roll out (Dialog Semi)

- **External training courses & workshops**
- **Internal seminars & knowledge sharing**
  - Best practice guidelines
  - Wiki knowledge base
  - Code examples
- **External OVM resources**
  - OVM Forum
  - Verification Academy
  - External consultants
- **Introduction on live projects**
  - Code review sessions
- **Library OVC components**

From VerificationFutures 2011
Reproduced with permission from Dialog Semi
Some conclusions (Dialog Semi)

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

• **OVM 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
Adoption: Get on board the Mentor UVM Express

• **Step #1 Organize your Testbench into a BFM**
  – Use a SystemVerilog Interface to group your Signals
  – Write your test in terms of transactions
  – Call tasks to execute transactions

• **Step #2 Add Functional Coverage**
  – Use Metrics to check Verification quality- How good are your tests?
  – Add coverage agents
  – Leverage pre-built VIP in passive mode

• **Step #3 Add Constrained Random Stimulus**
  – Improve your test quality by generating stimulus efficiently
  – Leverage pre-built VIP in active mode

• **Step #4 Use the full power of the UVM**
  – Modify your environment to improve reusability and configurability
  – Leverage all your code from the previous steps
Structure your teams like your test benches
But let’s put this in context

- It is not just about building great test benches!

- What are your signoff metrics?
  - And how do you track progress?
  - What are your milestones?

- What are your coverage scenarios?
  - What is your process of defining them?

- How do you measure checker quality?

- What is your VIP strategy?

- How do you integrate with formal?
The Importance of a Plan

Project Manager
*Tracking status*

Verification Plan

Architect
*Ensure intent is realised in design*

Verification Engineer
*Common status document & buy-in*

Verification Environment Development

Design Engineer
*Ensure implementation is in line with spec*

Execute Sessions

Coverage Metrics

Debug

From VerificationFutures 2011
Reproduced with permission from Dialog Semi
The mechanics of finding a bug

**Stimulate**

- ....01010101
- ....01001101
- ....10011010
- ....01001101

**Propagate**

- 01100101
- 11110101
- 00010101

**Design Under Test**

**Observe**
Why do we need VIPs?

- **Time To Market**
  - Ready-to-integrate models accelerate development

- **Quality**
  - Improve thoroughness of verification using VIP with pre-defined tests, coverage models, assertions, …
  - Demonstrate compliance to a protocol
  - Licensing (or buying) VIP imports knowledge

- **Reduce costs**
  - Increase re-use
    - Vertical: use VIP at block and SoC level
    - Horizontal: use VIP across multiple chips
    - Industry: External VIP should be cheaper to license (or buy) than make
  - Does VIP cost less to use than it for you to develop it?
Summary

• **UVM can do the job**
  – Of building constrained random environments

• **But it is not easy to learn or deploy**
  – Plan your ramp
  – Plan and monitor your adoption

• **It is NOT a silver bullet**
  – Keep it in context

• **UVM is a great step for our industry**