Managing Functional Verification Projects
Meeting the challenges of high-level verification in today’s SoCs

Kwamina Ewusie, Senior Consulting Manager, Synopsys Professional Services
Rajat Mohan, Product Marketing Manager, Synopsys Professional Services

October 2007
Introduction
The adoption of advanced verification languages and methodologies requires evolution of project management techniques in addition to the change in engineering practices. Managers must be able to assess and manage key project elements such as team expertise, verification methodology, verification IP (VIP) selection and environment setup to successfully deploy high-level verification environments. These factors take on increased significance for teams that are new to constrained random verification techniques and advanced languages such as SystemVerilog.

This paper not only explains how these key success factors affect verification productivity, but also provides recommendations for optimized deployment. Finally, there are recommendations to identify and resolve common bottlenecks that often impede a successful migration to advanced verification methodologies.

High-Level Verification Techniques
The increasing functional complexity of SoCs and the associated growth in design logic state-space has led design teams to adopt new languages and methodologies in their efforts to maintain design quality while improving design productivity. Specifically, advanced verification languages such as SystemVerilog, OpenVera, and others have gained popularity as they provide constructs to implement constrained-random testing as well as assertion-based verification. For the purposes of this white paper, the term “high-level verification” (HLV) refers to verification techniques that have the following characteristics:

• Constrained random testing (CRT)
  – Transaction-level stimulus modeling: High-level data structures (objects) encapsulate stimulus (akin to conventional test vectors) into transactions (high-level functionality such as read, retry, drop packet etc.) The testbench generates and processes these high-level transactions to increase efficiency compared to writing test vectors manually.
  – Random stimulus generation: Constraints describe the legal range and relationship among transactions (for example, a write operation must have addresses between 10 and 100). Solving these constraints at runtime provides pseudo-random stimulus generation.
  – On-the-fly response checking: The testbench contains a reference model of the design under test (DUT) and can therefore predict the response of the DUT. Design failures are determined at runtime.
  – Coverage driven: Functional coverage models describe all the features that need to be verified. As random tests are run, the coverage models provide a metric that can be used to track verification progress.

• Assertion-based verification
  – Assertion-based checkers/monitors: As defined by the test plan, assertion based monitors are deployed at the block and chip boundaries to capture intended behavior. Additionally, micro-architecture level assertions are used to define the designer’s assumptions that might not be part of the test plan.
  – Assertion-based coverage:Assertions are used to ensure that the stimulus exercises gray-box conditions within the RTL. Coverage of these assertions provides a measurement of gray-box testing.

Adoption of high-level verification poses both technical and non-technical challenges, especially for teams that normally use directed test methods. This paper focuses on the non-technical challenges. The goal is not to provide a comprehensive checklist of tasks involved in a high-level verification project, but rather to articulate some of the key differences between HLV and traditional directed test verification. Understanding these differences will help design and verification teams better manage the adoption of HLV techniques.
Key Success Factors for Advanced Functional Verification

A successful implementation of constrained-random verification requires understanding of several key success factors, including:

- Team expertise
- HLV language and methodology
- Verification IP
- Regression environment

Team Expertise

Architecting and implementing a CRT verification environment is a process that is more akin to software engineering than hardware engineering. Thus, it is important for the team to include at least one engineer with a strong object-oriented software programming (OOP) background to guide the development of testbench architecture. A software background will help in using OOP semantics of the HLV language to architect testbench components for reuse and scalability.

Furthermore, for designs that have complicated transaction types, algorithms common in the software engineering domain can be applied to verification very effectively. A binary tree algorithm can be used to manage search and sort operations, for example, and hash arrays can be used to model memories.

As with directed verification, a key ingredient for CRT teams is domain expertise. The difference in CRT is that a high level of abstraction supports the ability to encapsulate low-level details from the end users. Thus, the domain expert can create an abstracted environment that other verification engineers (with limited domain experts) can use to complete functional verification tasks. The availability of assertions and self-checking protocol monitors also facilitate the debugging process, especially for team members with limited domain expertise. Thus, CRT's higher level of abstraction helps minimize the bottlenecks that might otherwise be present due to the lack of domain expertise.

HLV Language and Methodology

Compared to other HLV languages, SystemVerilog is gaining popularity since becoming an IEEE standard. All major tools including simulators, synthesis, linters, etc. now support SystemVerilog. For a verification project, besides the extent of language support, the ability and efficiency of the HLV simulator's constraint solver to solve complex constraints becomes a critical component of tool selection. Another advantage that an HLV like SystemVerilog provides is a range of constructs that simplify the task of implementing high level software-like algorithms.

What may be less evident, but equally important, is the need for a proven implementation methodology for the chosen HLV language. Unfortunately, it is common for a verification team to adopt an HLV but use its syntax to write testbenches in a manner similar to traditional Verilog testbenches. This approach fails to reap the benefits of the HLV. The VMM verification methodology helps avoid this trap. This methodology, defined in the Verification Methodology Manual for SystemVerilog, captures best practices and recommendations for developing a more efficient verification environment and increasing the likelihood of first pass silicon success. It also provides a rich base-class library that boosts the testbench implementation team's productivity and helps team members understand how to use the object-oriented features of SystemVerilog effectively.

Verification IP

Using proven verification IP (VIP) can significantly increase the productivity of the verification team by reducing the testbench development time. A verification team may choose to develop VIP for internal use/reuse or acquire VIP from external sources. Regardless of the build/buy decision, it is highly recommended that the verification team use VIP for standard protocols, as high-quality VIP can ensure interface compatibility and compliance with protocols and interfaces.

It is important to consider how the VIP can be integrated within the chosen methodology. For example, the VIP should provide a transaction level interface to the verification environment. It should have native coverage models to gather and control functional coverage.
Table 1: Build vs buy decision for verification IP

Regression Environment
An HLV-based regression environment must have the following features that may not be essential for directed testing:

- Support for executing a single test with multiple random seeds to create multiple test cases. The key is to ensure that only the seed for random generator changes between consecutive test runs.
- Ability to gather and merge functional-coverage metrics (besides code coverage) to generate reports for measuring verification progress and completeness, in addition to traceability from coverage to test case.
- Ability to reproduce a failing random simulation for the purpose of debug (random stability).
- Adequate disk space and memory to ensure that the simulations compile and run. An HLV-based flow generates an order of magnitude more runtime data (log files, coverage data, runtime data, etc.) than a directed flow. Tests might run longer since a pseudo-random environment is limited only by storage requirement besides time limitations.

Execution
When executing an HLV-based test plan, the following factors help to ensure measurable progress and high-quality results.

Review Team
Similar to the design review team, a verification review team helps improve the quality of functional verification. An effective verification review team has members from all functional domains involved in the SoC/system design process (Figure 1). It is also useful to have a champion who bridges the mindsets of the design and verification teams. For example, this person can help convince designers on the value of integrating assertions while writing the RTL. Designers should write micro-architecture related assertions. These assertions provide visibility into the design and are very helpful in testbench integration and system-level debugging.
The verification review team should meet at significant stages of the verification process, such as during test-plan creation, environment development, scenario generation, test debug, and regression planning.

Here is an example of how the review team can help in debugging a test failure: Random combinations of legal constraints on register values can lead to illegal configuration sequences causing test failures. This may be due to an ambiguity between specification and implementation. The system architects in conjunction with the software lead can help define the acceptable behavior and remove ambiguity. Without the review team in place, such an issue would take much longer to resolve.

**Learning Curve**

As mentioned earlier, an HLV flow requires verification engineers to learn more than just syntax, because going from Verilog to HLV coding styles requires a shift in mindset. Verification engineers need to learn software-like flows that take full advantage of object-oriented programming (OOP) capabilities of HLV languages. To jumpstart the learning process, it helps to go through language training first. Then after a few weeks of hands-on experience, the engineers can go through customized methodology training. Adopting assertions on existing directed environment is another approach that helps teams gradually migrate to a full HLV flow.

Most successful migrations to HLV-based flows have used a tiered team structure. Testbench architects with OOP/software background formulate the strategy and flow. Verification engineers then work in a well-defined framework to build modular, reusable testbenches. Standard methodologies such as VMM help to implement and enforce a consistent verification flow.

**Phased Execution**

Figure 2 shows the timeline for traditional and constrained random HLV environments, using a sample project with two RTL releases (Rev_1 and Rev_2). Note that a comprehensive CRT environment does take longer to architect and implement than a directed testbench, where unit-level testing may begin a few days after the start of the verification project. The advantage of investing in a comprehensive environment up-front is the large gain in productivity throughout the overall verification effort (i.e., functional coverage attained in a given amount of time).

To help achieve seamless migration from a directed to an HLV flow, the latter should include bring-up testing (aka “sanity test”) capabilities at an early phase of testbench development. This testing helps the design team to bring up the RTL efficiently and more easily adopt the new verification flow as they don’t have to write extensive standalone testbenches for standalone unit level testing.

Furthermore, the functional coverage model must be written as part of the testbench development phase. When the random test environment is ready to run, the progress of the verification effort can be analyzed by collecting functional and code coverage data. Automation of coverage analysis and mapping to the test plan is essential to make hundreds of megabytes of raw coverage data intelligible.
Execution Bottlenecks

In a typical verification project, where a larger percentage of total verification cycles are spent running and debugging test cases, a methodology that simplifies debugging has a major impact on productivity. Techniques such as assertions and unified message logging aid in debugging.

A robust test plan should have a comprehensive definition of the functional coverage model as well. Since the number of distinct tests developed and run is not a good measure of progress in CRT-based flows, it is essential to implement and collect functional coverage data. In order to successfully deploy functional coverage, it is important to determine the impact of adding coverage points on the overall simulation performance. Adding too many assertions or functional coverage points to insignificant design nodes adds unnecessary simulation overhead. Similarly advanced coverage features such as cross-binning allow you to generate interesting results, but if not used judiciously will have adverse effect on productivity. For recommendations on best practices to optimize simulation performance, please see the references section at the end of the paper.

Verifying the registers of a highly configurable design consumes lots of CPU cycles. Also, as most verification managers might have seen, the register definitions change often and sometimes significantly. The representation of the registers at a higher level of abstraction increases productivity by allowing efficient data manipulation. Applications like register abstraction layer (RAL) that are part of VMM provide an efficient means implement this functionality.

All of these capabilities can be put into place by applying a good HLV methodology like VMM hand-in-hand with appropriate tools. When making these methodology and tool decisions, managers must understand that there will be a learning curve in adopting the HLV methodology, especially if moving from a directed test flow. But the advantage is an overall gain in productivity and potentially a significant gain depending on the complexity of the design. Optimal utilization of resources can be achieved by deploying off-the-shelf tools and solutions with appropriate customization.
Summary
From a project management perspective, there are significant differences in processes and success factors in traditional directed verification flows compared to HLV-based verification flows. The key issues include technical expertise, infrastructure requirements, IP considerations, and the metrics for measuring verification progress. Being aware of and planning for these success factors will help engineering managers take full advantage of the productivity gains afforded by HLV methodologies.

References
1. Interviews with Synopsys Professional Services consultants with hands on experience in verifying numerous SoCs for various applications.
3. Verification Methodology Manual for SystemVerilog — from Synopsys and ARM
6. SystemVerilog Assertions are for Design Engineers Too! — SNUG Europe 2006
8. Complex Register Verification Utilizing RVM Based Register Abstraction Layer (RAL) — SNUG Boston 2006
9. RAL Demystified: A Fast Path to Smart Control Plane Verification — SNUG India 2007

Abbreviations
• CCRT — Constrained Random Testing
• DUT — Design Under Test
• OOP — Object Oriented Programming
• SoC — System on Chip
• SVA — SystemVerilog Assertion
• VIP — Verification Intellectual Property
• VMM — Verification Methodology Manual
• VMM-RAL — Register Abstraction Layer, a VMM application
Author Biography

Kwamina Ewusie
Senior Consulting Manager, Synopsys Professional Services

Kwamina Ewusie is a Senior Consulting Manager for Synopsys Professional Services. He joined Synopsys in 1998 as a professional services consultant and has delivered on several design verification and implementation projects. He currently manages verification and implementation projects for customers in the south west USA, from his office in San Diego, CA. Kwamina has experience in embedded software development and design experience in DSP, microcontroller and SoC design domains. He has an MEng (Hons) from University of Manchester Institute of Science and Technology (UMIST), Manchester, England.

Rajat Mohan
Product Marketing Manager, Synopsys Professional Services

Rajat Mohan is a Product Marketing Manager responsible for promoting Synopsys Professional Services business. He joined Synopsys in 2003 as an applications consultant supporting functional verification products. He has over 10 years of experience working on designs ranging from DSP, embedded processor and networking applications. He has worked at Motorola, Nortel, CoWare and several start-up companies. He has a MBA from Santa Clara University and a BS (Hons) Electrical Engineering from Punjab Engineering College, India.

About Synopsys Professional Services

Synopsys Professional Services provides a broad range of consulting and design services to chip developers worldwide to help them achieve success in their design programs. These services address all critical phases of the SoC development process and are tightly aligned with Synopsys’ EDA tools and IP products to help customers accelerate their learning curves, develop and deploy advanced methodologies, and achieve successful tape-outs. We offer customers a variety of engagement models to address their project-specific and long-term design needs. For more information on Synopsys Professional Services visit our website at www.synopsys.com/sps.