Risk reduction in a verification upgrade

Ambar Sarkar, Paradigm Works
Brian Bailey, Poseidon Systems

Abstract

Product managers are under enormous pressures. Designs are getting larger and more complex, and in addition managers have been barraged by a massive upgrade cycle in the verification side of the team. New methodologies (VMM, RVM, eRM), languages (PSL, e, OVL, SystemVerilog, SystemC) and tools have been released, with each vendor claiming that they know how to increase quality and reduce the risk associated with getting that chip out of the door on time and fully working. With any methodology change, there are inherent short and long term risks as the team learns new techniques. Managers need a process by which they can evaluate possible changes, decide which ones are right for them to adopt, perform the insertion into the team and to measure the effectiveness of the changes. This paper will examine the process along with some specific examples related to the adoption of the recently released Verification Methodology Manual for SystemVerilog[1].

Introduction

Designs today often include multiple heterogeneous processors with complex hardware / software interfaces, increased levels of concurrency and large quantities of reuse. This increase in design complexity leads to an even bigger increase in verification complexity, and with verification now consuming on average 70% of the total time, designers must find ways to make the verification processes more efficient. One way to do this is with verification reuse, but for this to be effective and to enable external verification IP to be utilized there must be a methodology that allows the industry to build these verification components into a complete verification flow. A recent example of this, the Verification Methodology Manual for SystemVerilog (VMM)[1], attempts to define rules for a verification methodology such that industry wide verification reuse will become possible due to the standardized way in which each of the components has been created. The VMM is based on years of experience, not only with SystemVerilog, but also with previous languages such as e and Vera.

Verification Upgrade Cycle

One way to improve productivity is to increase the level of abstraction. With the introduction of new languages, such as Vera and e, and more recently SystemVerilog, a quiet revolution started in verification. This is because the languages encouraged a verification engineer to model the environment and its interaction with the design rather than just creating stimulus sets, or in other words to define the system from the outside looking in. These are often called constraints. This enables stimulus sets to be created automatically by a tool rather than by hand. With the introduction of this methodology,

two other needs were exposed: the ability to calculate the expected responses automatically, and the ability to record functionality that had already been verified such it could alter the behavior of the test case generator. An additional set of languages constructs allow aspects of a design to be described declaratively rather than procedurally. These constructs form the basis on which assertions are built. With these new concepts in place, the elements of a modern testbench were born. But a language is only half of the solution, it has to be applied properly to maximize its value, and to assist with this a number of verification methodologies have been proposed such as the e Reuse Methodology (eRM)[2], the Reference Verification Methodology (RVM)[3] and the newly released VMM.

The Verification Methodology Manual for SystemVerilog

The primary objectives of the VMM are:

1. to ensure a high quality verification environment,
2. to enable reuse of verification components,
3. to enable testbenches to be assembled out of reusable blocks,
4. to ensure that verification IP is of a consistent quality,
5. to encourage a common vocabulary among verification engineers.

The VMM is organized into a number of discrete sections, such as Verification planning, Assertions, Testbench Infrastrucure etc. While each of these is somewhat standalone, they are designed to enable complete systems and flows to be assembled. In addition to the methodology, a class library is defined which is currently provided by Synopsys to customers of its VCS simulator. It provides most of the necessary infrastructure capabilities and ensures that information can flow between the pieces effectively. The VMM also defines an expanded set of assertions that builds on the Accellera Open Verification Library (OVL)[4].

But it is not easy to pick up the VMM book and to start using it. It is new and tool support for it is provided only by one vendor today. Undoubtedly other implementations of the class library will emerge, and Synopsys plans to add support for other vendors' simulators[5]. Even if you are not ready to adopt the SystemVerilog language today, it is still possible to extract a lot of wisdom from this and the other verification methodologies. They can help engineers create better verification environments that will be easier to re-use in the future.

Risk assessment

With change there is risk. Engineers need to learn new techniques and in many cases give up the ones in which they have years of experience. This may cause a reluctance to change. But it is equally risky not to make changes. In a DAC 2003 paper[6] Intel published some results about bugs. They stated that the total number of detected bugs increases 3 to 4 times for each generation of the IA-32 architecture and projected that in their next generation they expected to find 25,000 bugs. It should come as no surprise that processor vendors are the leaders in the adoption of new verification methodologies. They can thus provide guidance for where other companies should start. Assertions have been in use by these companies for a number of years and they have all reported positive experiences with their adoption. As examples, 34% of all bugs found were identified by

assertions on the DEC Alpha 21164 project[7]. At HP 85% of all bugs were found using over 4000 assertions[8] and 400 bugs were detected by Intel using formal proofs of assertions[6].

Managing Risk

Unfortunately, design managers cannot afford to wait for conclusive evidence before looking at adopting these new verification methodologies. That makes it important to have in place a good policy for managing and minimizing the risks associated with adoption and for being able to assess the impact that each change will have, both positive and negative. The complete process is iterative, so once it has been started, the collected data will accumulate over time making future changes easier to assess and for the changes to be made in response to particular situations.

The process has four main steps: evaluate, select, insert, and monitor, which are described below.

Evaluate situation

No engineering team is completely happy with their design and verification flow. Often the current design will expose weaknesses that need to be improved in the future. It is important to know why change is being made and the expected results of it. One of the big problems with introducing new techniques is a lack of experience with them. Consider the adoption of the VMM. It requires knowledge of object-oriented programming, declarative statements and the correct way to structure test benches. Do you have the necessary skills within the team and if not how are you going to get them?

Assume that you have decided to adopt the VMM methodology. You must keep in mind that while most of the rules are meant to increase the overall efficiency of the project, each rule often takes incremental effort. For example, VMM recommends putting in hooks for adding callback methods that are invoked before and after significant actions are taken by a transactor. Implementing these callback hooks means that you can add interesting error stimuli by making changes to the transaction right before it is driven to the DUT without modifying the original transactor code. If no callback methods are eventually needed, one need not have spent the extra time needed to define various callback prototypes.

The adoption of the complete methodology in one single step is impractical in the authors' opinion. There are too many rules, and not all rules are applicable to all projects. Second, by their shear number, you cannot expect a verification engineer to understand and conscientiously apply all the rules. Third, there are no tools to enforce these rules and check for violations. In fact, it is highly unlikely that there will ever be such a tool since for many of the rules there is no way to check compliance. The authors thus recommend an iterative adoption process. The team should identify and agree on a subset that it expects team members to understand and follow in a reasonable amount of time.

Steps to be undertaken

- Identify the goal. What do you hope to improve or how would you measure success? For example is verification re-use the goal?
- Identify the rule set that needs to be adopted. Make sure the team understands why these rules are being selected.
- Establish the skills necessary for success. Don't forget to look at related groups like the software team for some of these skills, such as object-oriented programming.
- Identify areas of weakness and put in place a plan to deal with training. EDA vendors and consultants have offerings available to help with the VMM.

Select upgrade

With the goal determined, a decision has to be made about what to change and to analyze the impacts that this may have on other aspects of the flow. For example, if you decide to adopt pseudo random test generation have you properly considered that additional verification engineers may be required early in the development process, rather than after the RTL implementation has been completed as is the case with older verification methodologies? Or that with the increased ability to generate large quantities of test cases, the number of simulator licenses or machines available to run them on may not be sufficient?

Adopting VMM standard library

There are about 10 base classes available in the standard VMM library. These classes cover messaging and output (vmm_log), simulation execution flow (vmm_env), layered testbench components (vmm_xactor, vmm_atomic_gen, vmm_scenario_gen), communication objects (vmm_data, vmm_notify, vmm_channel, vmm_scheduler, vmm_broadcaster). Each of these classes captures the basic functions that need to exist in any verification environment, and are made fairly flexible by virtue of object oriented principles. For example, vmm_log is fairly flexible and powerful implementation of messaging, and while it seems like overkill, spending time here can make a difference in the convenience of debugging verification output.

The authors recommend that you start by adopting the following subset: vmm_log, vmm_env, vmm_xactor, vmm_data, vmm_channel, vmm_notify, and vmm_atomic_gen. These represent a subset for the most basic implementations of the VMM testbench. Once you have become comfortable with these classes, it is an incremental effort to incorporate the remaining base classes related to advance communication (vmm_scheduler, vmm_broadcaster) and stimulus generation (vmm_scenario_gen).

Implementing non VMM base classes

While the base classes offer a great start, there are a couple of classes that need to be implemented and are absent from the VMM library. Scoreboards are talked about extensively but it does not provide a base class implementation. Similarly, while some recommendations are provided regarding how to bring the simulation to an end, a utility class to implement shutdown may be a good addition.

Steps to be undertaken

- What is the impact on the complete flow that the adoption of the chosen rule set will have? Ensure that people, tool and machine issues are addressed.
- Decide on build versus buy. Creating your own base class implementations will give you vendor independence, but is that a good use of your time?
- Are verification components going to be purchased? What impact will that have on other decisions - what means of integration do they provide?

Insertion

Many companies like to do a pilot project first. This may restrict the change to a small part of the design and provides a period when the new techniques can be learned and perfected by a subset of the team. Next time around they are able to help the rest of the team adopt the new technique.

Suggestions for places to start

Verification plan

Switching to a constrained random and functional coverage driven verification approach requires changes to a traditional verification plan, and the VMM devotes a chapter to this topic. Instead of specifying how to create a directed test case, the key focus is now how to constrain random stimuli, identify useful stimulus sequences and data sampling interfaces, and associating features with their corresponding functional coverage models. Almost all of the rules in this section are candidates for adoption[9].

VMM recommended testbench architecture

The VMM devotes a chapter to testbench architecture. Before starting to read this chapter, it is important to be familiar with object-oriented programming principles. Another key concept is to understand how the class-factory pattern works.

While all of the rules and guidelines specified in that chapter are useful, many of them are already implemented in the base class implementations as defined in the VMM library. Rules that recommend that tests be included in program blocks to avoid race between design and verification code should be followed diligently as not doing so may lead to hard to debug erratic behavior by the verification environment. Rules that describe how to use callbacks to integrate scoreboards

and functional coverage can be deferred as it often depends on the nature of the functional coverage information being collected.

Exception handling

Verification of DUT exceptions is tricky and often hard to implement if not planned for. In such cases, it may require significant changes to the existing code. Callbacks offer a solution to this problem. VMM recommends how to implement these hooks in transactor code so that user-defined methods are called before and after major activities such as driving signals to the DUT or detecting major events. This allows the user to inject and check for errors without requiring changes to the existing code. As a consequence, the authors recommend that the user get familiar with these callbacks.

Understanding how to incorporate legacy verification components

While it is often easier to adopt a new methodology by starting on a project from scratch, in reality this is not often the case. Reuse of transactors that have been developed using different methodologies or different languages will be helped by the guidelines in Chapter 4, section "Legacy Bus-Functional Models". This provides guidelines for creating wrappers which upgrade the legacy models for use in a VMM environment.

Adopting assertions

Assertions are becoming an essential tool for isolating bugs and implementing functional coverage, but many of the suggested guidelines will not be easy to understand. The authors recommend that the team start by using the checkers in the VMM Library. While the design team uses these checkers, the verification team can either create more sophisticated assertions, or extend the VMM library. Using such an approach will help the team adopt assertions without having to invest much of the team's time in learning SystemVerilog Assertions (SVA).

Specifying constraints

While the specification of constraints in SystemVerilog is not too difficult, writing them in an efficient manner is critical. The authors recommend that the team spend time understanding the section titled "Data and Transactions" in chapter 4. In any project, some amount of time is spent in specifying and debugging the constraints, and the time spent in understanding these rules and regulations will be useful.

Adoption of System-level and Processor-integration Verification.

The authors recommend deferring the adoption of system-level and processor integration verification until the concepts in chapters 1-7 have been fully mastered. It is also the authors' opinion that a significant amount of System-Level Verification can already be accomplished by the approaches specified in these earlier chapters. The adoption of processor integration verification can be introduced in a totally separate project.

Monitor effectiveness

When any new technique is added, its impact must be fairly evaluated. The authors recommend more code reviews than normal allowing everyone to learn from each other - spread good ideas and stamp out bad practices early on. It is also important to provide some level of “control” so that effectiveness can be measured. Coverage metrics may provide a way to do this. Compare progress of functional coverage against code coverage. Use this to assess the quality of the new metrics but also to point to the weak spots of the old. This will also give added confidence in later iterations as the "shape" of progress in the new metrics becomes more accepted.

Conclusion

The VMM is large with many component pieces and it is not necessary to adopt it all at once. In fact doing so without understanding why may be quite risky. By carefully selecting an order of adoption and taking it in steps, the necessary experience can be built into the team to ensure that adoption is successful. Doing it all at once may lead to frustration and failure.

Reference

[1] Bergeron, Cerny, Hunter, Nightingale: *Verification Methodology Manual for SystemVerilog:* Springer 2005.

[2] Verisity website: http://www.verisity.com/products/erm.html

[3] Synopsys website: http://www.synopsys.com/products/simulation/simulation.html

[4] Accellera OVL Committee Homepage: http://www.accellera.org/activities/ovl

[5] http://www.synopsys.com/news/announce/press2005/snps_sourcode_licsvpr.html

[6] Tom Schubert: "High Level Formal Verification of Next-Generation Microprocessors". DAC 2003

[7] Kantrowitz and Noack: " I'm done simulating; now what? Verification coverage analysis and correctness checking of the DEC 21164 Alpha microprocessor" DAC 1996

[8] Foster and Coelho: "Assertions Targeting a Diverse Set of Verification Tools " HDLCon 2001

[9] Ambar Sarkar, “Creating a VMM Compliant Verification Plan”, SNUG San Jose 2006

The Authors

Dr Ambar Sarkar is a Principal Consulting Engineer with Paradigm Works, a premier verification and development services company with a worldwide client base.

Brian Bailey is the Chief Technologist of Poseidon Design Systems as well as a Technical Advisory Board member and consultant to a number of other EDA companies.