Reusing Verification Components in System-Level Modeling Environments

Ambar Sarkar, Jim Crocker
Paradigm Works, Inc. MA, USA
{ambar.sarkar|jim.crocker}@paradigm-works.com

Bob Ionta
The MathWorks, Inc, MA, USA
bob.ionta@mathworks.com.

Introduction

As the specification of a system becomes increasingly complex, the task of verifying its implementation becomes exponentially harder. In the domain of functional verification of digital hardware, most state of the art verification environments have evolved into complex distributed software architectures, composed of a number of verification components such as stimulus generators, response checkers, monitors etc. In an attempt to create realistic execution scenarios for the system, the verification engineer executes a model of the system along with these verification components, each executing concurrently and collectively exhibiting complex interaction patterns. Several hardware verification languages such as Vera, e, SystemC, and SystemVerilog[1,2,34] have emerged to describe these complex verification components. In addition, advanced verification methodologies such as VMM, AVM, and eRM[5,6,7] have been developed by EDA tool vendors and the user community to create such environments efficiently and in a reusable manner.

Complementary to the verification effort, system architects and implementers require means of communicating requirements and specifications that is reliable and understandable at the level of whole-system behavior as well as detailed implementation levels. The inadequacy of paper based specifications leads system designers to prefer to communicate via executable specifications such as Simulink® models[8]. The entire system can be modeled and simulated using Simulink®, thus lending provable accuracy and reliability to the communication. Refinements to the model can incorporate implementation specific parameters into the simulations and validate the implementation in the system context with system level stimuli. The same mechanism that allows addition of bit-true and cycle-accurate subsystem implementations to the system model can be used to incorporate detailed transaction level models as well, providing the ability to thoroughly validate protocols in context and with the controllability and features those models provide.

In this paper, we present a novel approach of taking advantage of an emerging verification language and reuse-based verification methodologies to reduce the complexity of the tasks of both functional verification as well as system-level modeling. Specifically, we show how to use SystemVerilog, a hardware verification language (HVL) recently standardized by IEEE, to model a system at various levels of abstraction. Next, we show how we take advantage of emerging verification methodologies such as AVM and VMM to define standard interfaces for verification components. We then present a case study that illustrates how an AVM-compliant verification component can be used to generate more accurate transaction-level stimulus in system-level modeling environments such as Simulink®.

SystemVerilog – A Hardware Verification Language

Verification of a complex system places unique demands on the language used for verification. The first that comes to mind is the ability to perform transaction level modeling of the interaction between the design and the environment. In addition to speeding up the simulation, having transaction-level view of the various interactions within the system helps more efficient debugging.

Another requirement of a HVL is to model the environment and its interaction with the design rather than just creating stimulus sets manually. For example, automatic creation of rich stimulus scenarios can be possible if there are language constructs to specify constrained-random parameters for the stimulus generating code. This enables the creation of stimulus sets automatically by a tool rather than by hand.

An additional set of language constructs should allow aspects of a design to be described declaratively rather than procedurally. These constructs are known as assertions, which can describe the complex temporal behaviors of the system concisely and unambiguously. Finally, the language should provide a means to quantify functional coverage, Functional coverage specifies the degree to which the verification task for the system has been accomplished.

Interestingly, all of these features of an HVL can and should be used for system-level modeling as well.

The remainder of this section provides examples of the key features of SystemVerilog that can be very useful in a system-level modeling environment such as Simulink®.

Advanced Verification Methodologies

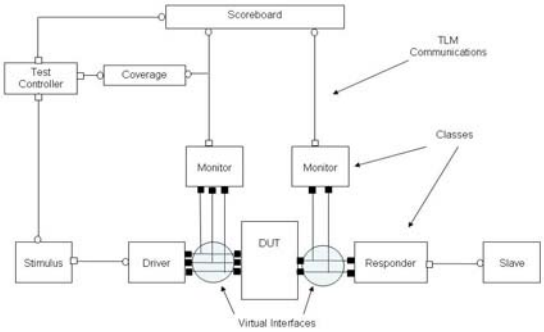

Figure 1: AVM Based Verification Environment

Another important reason for increased complexity in the verification environment is the fact that many of the verification components may be developed externally by third party vendors, or are available as legacy. Whenever feasible, project managers would like to reuse these components to avoid the cost and schedule pressure of developing them from scratch. It is often a big challenge trying to understand how these externally developed models work, let alone try to make these models work together. To address this concern, the EDA developers and the user community have collaborated to come up with several powerful verification methodologies, whose primary objectives are:

- to ensure a high quality verification environment,
- to enable reuse of verification components,
- to enable testbenches to be assembled out of reusable blocks,
- to ensure that verification IP is of a consistent quality,
- to encourage a common vocabulary among verification engineers

There are three specific ways these verification methodologies accomplish these objectives:

a. By codifying the flow of a simulation session. These methodologies specify an order in which all the components are instantiated, randomized, activated, and shut down,

b. By standardizing the interface of the verification components, and using object-oriented approach to refine their behavior, and,

c. By providing a number of guidelines that should be followed for creating methodology compliant models.

Figure 1 above shows a typical verification environment based on AVM, an advanced verification methodology supported by a major EDA tool vendor.

Integrating VMM-compliant components in System-level Modeling Environments

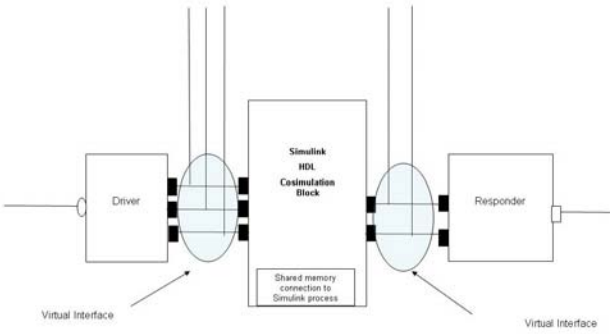

Figure 2: Integrating AVM Compliant Verification Environment With Simulink®

Figure 2 shows how a AVM based verification component can be integrated within a Simulink® based system-level modeling environment. Instead of using a Verilog implementation of the design, one can replace it with a Verilog wrapper, called the Simulink HDL cosimulation block in the diagram. This wrapper allows the communication of both stimulus and responses between the verification environment and a concurrent Simulink® model execution session. As a result, the Simulink® modeling environment can take advantage of the rich stimulus generation and even response checking offered by the verification environment.

References

[1] OpenVera Language Reference Manual: Testbench, Version 1.4.3, September 2005

[2] IEEE 1647, The e Functional Verification Language Working Group, http://www.ieee1647.org

[3] OSCI WG Verification: SystemC Verification Standard Specification V1.0p1. http://www.systemc.org.

[4] IEEE P1800, "Standard for SystemVerilog Unified Hardware Design, Specification and Verification Language," IEEE, 2005

[5] Verification Methdology Manual for SystemVerilog, Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale, Springer, 2005

[6] Verification Cookbook, AVM Library Documentation, Mentor Graphics, 2006

[7] eRM – e Reuse Methodology, http://www.verisity.com/products/erm.html

[8] Simulink® –Simulation and Model-based Design http://www.mathworks.com/products/simulink