Introduction to the New Accellera Open Verification Library

Harry Foster
Jasper Design Automation, Inc.
Mountain View, CA, USA
harry@jasper-da.com

Kenneth Larsen
Mentor Graphics, Inc.
Wilsonville, OR, USA
Kenneth_Larsen@mentor.com

Mike Turpin
ARM, Ltd.
Cambridge, UK
Mike.Turpin@arm.com

ABSTRACT
The new Accellera Open Verification Library (OVL) standard fulfills the long-anticipated vision of creating a vendor- and language-independent assertion library that can be leveraged across multiple verification processes. In this paper, we present the new Accellera OVL standard, which is a completely re-architected version of the original OVL that takes advantage of new features offered by the emerging PSL and SystemVerilog standards.

Keywords
Assertion, Coverage, Constraint, Formal Verification, Simulation.

1. INTRODUCTION
The Open Verification Library (OVL) was first introduced to the world five years ago at the 2001 International HDL Conference (the predecessor to today's DVCon) [1]. Prior to the OVL, the industry lacked standards for specifying assertions. Instead, many proprietary assertion languages had emerged that made it difficult to reuse the assertion specification within a flow. But with the advent of the OVL, the industry saw its first alternative solution that enabled the engineer to "specify once" and then leverage assertion specification over multiple verification processes—such as traditional simulation and dynamic- and static-formal verification tools. Since that time, a revolution has occurred in the design and verification community, one that has seen the emergence of assertion-based verification (ABV) [2] coupled with the new powerful standards, PSL (IEEE-1850) [3] and SystemVerilog (IEEE-1800) [4].

In this paper, we present the new Accellera OVL standard [5], which is a completely re-architected version of the original OVL that takes advantage of new features offered by the fledgling PSL and SystemVerilog standards. We discuss the OVL standard's new features and present a roadmap for feature enhancements. Finally, we present examples of how to unify assertion specification in a flow by using the OVL combined with PSL and SystemVerilog.

1.1 What Is the OVL?
The OVL provides designers, integrators, and verification engineers with a single, vendor- and language-independent template interface for design validation. However, the OVL is actually a *library* of predefined assertions, not a temporal property or assertion language (such as PSL or SVA). In reality, the OVL is a library-based assertion methodology that lets the engineer use the same assertion specification with different languages (which means good tool support across different flows) and in different parts of the design flow (for example, RTL simulation, formal proof, emulation, and FPGA prototyping). A designer can start adding assertions quickly (without taking on an appreciable learning curve or making mistakes in a language) and have confidence in the pre-defined assertions in OVL, which have undergone considerable testing.

1.2 The Value of OVL
While standardizing property and assertion languages is integral to addressing increased verification complexity, it is not the entire solution. Equally important to the ABV revolution is an effective methodology that unifies traditional and formal verification within an ABV framework. The new Accellera OVL provides a systematic element of an ABV methodology. For example, the OVL incorporates a consistent and systematic means of specifying RT-level implementation properties structurally through a common set of library elements (that is, assertion monitors). The OVL library elements act like a template that lets designers express a broad class of assertions in a common, familiar RTL format. Furthermore, the OVL capitalizes on the various emerging property and assertion language standards by unifying the PSL *declarative* form of property specification with the new SystemVerilog *procedural* form of specification within the library. In addition, these library elements address assertion-based methodology considerations by encapsulating a unified and systematic method of reporting that can be customized per project and a common mechanism for enabling and disabling assertions during the verification process. The reporting and enable/disable features use a consistent process that enforces uniformity and predictability in a project's assertion-based methodology. Finally, extensions in the new Accellera OVL provide an automatic means to measure coverage that enables verification teams to improve the efficacy of the test environment.

Fundamental to appreciating the benefits of assertions in general and OVL assertions in particular, is an understanding of the concepts of controllability and observability.

Controllability
Controllability refers to the ability to influence an embedded finite state-machine, structure, or specific line of code within the design by stimulating various input ports. While in theory a simulation testbench has high controllability of the design model's input ports during verification, it can have low controllability of an internal structure within the model.

Observability

In contrast, observability refers to the ability to observe the effects of a specific internal finite state-machine, structure, or stimulated line of code. Thus, a testbench generally has limited observability if it only observes the external ports of the design model (because the internal signals and structures are often hidden from the testbench).

To identify a design error using the testbench approach, the following conditions must hold (that is, evaluate true):

1. The testbench must generate proper input stimulus to activate (that is, sensitize) a bug.
2. The testbench must generate proper input stimulus to propagate all effects resulting from the bug to an output port.

It is possible, however, to set up a condition where the input stimulus activates a design error that does not propagate to an observable output port. In these cases, the first condition cited above applies; however, the second condition is absent.

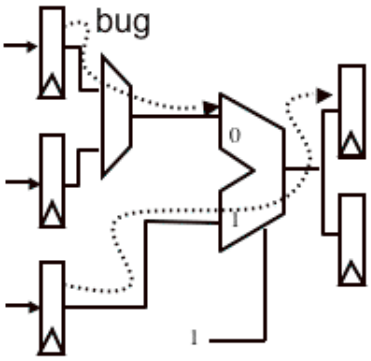

Figure 1. Poor Observability Misses Bugs

In fact, many project teams are amazed the first time they turn assertions on in their simulation environment and find that the tests that were previously passing, now fail due to improved observability.

Embedding assertions in the design model increases observability. And as a result, the verification environment no longer depends on the testbench generating proper input stimulus to propagate a bug to an observable port. Thus, any improper or unexpected behavior can be caught closer to the source of the bug in terms of both time and location in the design intent. In fact, experience demonstrates that it is not uncommon for a self-checking test to pass in the absence of embedded assertions, but then a previously undetected bug will be revealed as assertions are added to the design. Another surprising finding is that assertions can identify bugs in portions of the design that are either not directly related to or outside of the functional focus of a specific test. Meanwhile, an assertion within the functional focus of a test passes and satisfies a particular test's functional coverage targets.

OVL improves both Observability and Controllability

In general, assertions help solve the observability challenge in simulation, but they do not help with the controllability challenge. However, by adopting an assertion-based, constraint-driven simulation environment, or applying formal property checking techniques to the design assertions, we are able to address the controllability challenge. Furthermore, by including coverage measurements directly in an assertion module (such as the OVL), the verification environment can report valuable information that helps us uncover holes in input stimulus and missing scenarios (for example, cases where the stimulus never activated a triggering condition for a particular assertion) [11].

2. BACKGROUND

The concepts behind the OVL were originally conceived and published by Bening and Foster [6] at the Hewlett-Packard Richardson Texas Computer Technology Lab in the late 1990's. The goal of creating this library (predecessor to the OVL) was to find a solution that would solve the overhead cost of evaluating multiple formal verification tools that were emerging at that time. Unfortunately, due to the lack of standards, each tool had its own proprietary assertion language. Thus, before teams could evaluate tools, they had to take on the added burden of translating assertions to customize them for each tool. To address the cumbersome translation tasks, Bening and Foster created an assertion library consisting of a set of assertion checker modules. These modules allowed the assertion implementation details to be isolated from the functional intent of the check. As a consequence, the team was able to create multiple assertion libraries that were optimized for specific verification processes within a flow that used the same RTL-instantiated interface and semantics for a particular check. Bening and Foster found that it was significantly quicker to add a new assertion library, which encapsulated a proprietary assertion language, than change an entire design containing hundreds or thousands of assertions. In fact, they could create a new library in a matter of minutes versus the hours or days it took to update a design.

Aside from solving the multiple proprietary assertion language support challenge, Bening and Foster discovered that engineers actually preferred using the library approach over learning a new language. In addition, the library provided a means to enforce a common and unified methodology throughout the entire organization, while providing a way to add new assertion capability and features seamlessly throughout a project.

Ultimately, the concepts behind the Hewlett-Packard assertion library were leveraged to develop a first (non-standardized) version of the OVL, which Foster and Coelho published at the 2001 International HDL Conference [1]. This version of the OVL was donated to Accellera for standardization in 2002, and it resulted in the first Accellera standard, which was published in June of 2005 [5].

3. CURRENT WORK

In this section, we describe the structure of the Accellera OVL committee responsible for the standardization effort. We then highlight the issues that were addressed in the most recent release of the OVL standard.

3.1 Accellera OVL Committee Structure

The Accellera OVL is organized as one central committee and subdivided into multiple working groups that have expertise

in developing versions of the library to support different HDLs and property/assertion languages. The main OVL committee is responsible for ensuring that semantic consistency is preserved between the multiple language versions of the library. The committee consists of leading experts from EDA and the electron industry and operates under full visibility and openness, as required by an Accellera organization.

3.1.1 Verilog / SVA Working Group

The OVL-VSVA working group is chartered with defining and delivering the Accellera Standard OVL LRM and libraries of assertion checkers implemented in Verilog and SystemVerilog. In compliance with their charter, the working group released the initial 1.0 version in June 2005. The driving force behind the working group's continuing effort includes the following goals and objectives:

a) ensure that the specification of existing and new OVL checkers and interfaces are consistent and clear,

b) that the implementations in Verilog and SystemVerilog are consistent with the specification and support simulation, emulation and formal verification engines,

c) to resolve all know issues and issues found, and

d) to continue the work and evolution of future releases of Accellera Standard OVL.

3.1.2 PSL Working Group

The OVL-PSL working group is chartered with defining and implementing a PSL version of the library. A preliminary PSL (Verilog flavor) version of the OVL is now available for review in the latest 1.5 release. The current plan is to release the new PSL (VHDL flavor) version of the OVL in the 2.0 release, which is targeted for the summer of 2006.

3.1.3 VHDL Working Group

The OVL-VHDL working group is chartered with defining and implementing a VHDL version of the library. The current plan is to release the new VHDL version of the library in the 2.0 release, targeted for the summer of 2006.

3.2 OVL Issues

The OVL committee began its work on the standardization effort by collecting a list of issues and recommended enhancements they would consider as they developed the new version of the library. The OVL committee uses Accellera's Mantis bug tracking system to capture and track all reported issues and requested enhancements. You can access the OVL issues list under the Mantis project "*Std OVL Errata*" at http://www.eda.org/mantis.

3.2.1 Corrections

The committee collected more than 40 issues. They analyzed the issues and constructed the new version of the library accordingly. The committee considered backward compatibility as they updated the interfaces to the checkers to include finer control for handling and consistency throughout the library. The corrected errors ranged from simple variable name misspellings (for example, antecendent_expr to antecedent_expr in assert_implication) and semantic differences between simulation and formal verification to corner-case concurrency bugs within a few of the more complex checkers.

3.2.2 Clarifications

Some issues the committee addressed were not related to a logic error in a particular checker—but reflected semantic confusion described in the original library reference manual. Defining what should occur while a check is pending (as seen in n-cycle checkers such as "*assert_time"*) is an example of how the committee clarified and strengthened the specification. Another example includes clarifying when to pipeline checks (as in the *"assert_cycle_sequence"* checker). The committee's work resulted in a new OVL reference manual that includes waveform diagrams and simplified examples for all checkers in the library.

4. NEW OVL OVERVIEW

In this section, we discuss the details and new features of the new OVL implementation.

4.1 Interface Semantics vs. Implementation

The OVL was originally conceived as a set of assertion templates, where the semantics associated with a particular interface syntax (for example, the module name) are separated from how the checker might be implemented (for example, the Verilog procedural code of the check). This separation makes it possible to create different implementations that are optimized for a particular verification process within the flow—as long as the specific checker implementation does not violate its interface's semantic definition.

The current implementation of the OVL also embraces this philosophy. For example, in Figure 2 we demonstrate the new OVL assert_always checker (whose semantics specify that its Boolean test_expr argument is always true at every clock cycle).

```
`include "std_ovl_defines.h"

`module assert_always (clk, reset_n, test_expr);
    parameter severity_level = `OVL_ERROR;
    parameter property_type = `OVL_ASSERT;
    parameter msg="VIOLATION";
    parameter coverage_level = `OVL_COVER_ALL;
    input       clk, reset_n, test_expr;

`ifdef OVL_VERILOG
  `include "./vlog95/assert_always_logic.v"
`endif // OVL_VERILOG

`ifdef OVL_SVA
  `include "./sva31a/assert_always_logic.sv"
`endif // OVL_SVA

`ifdef OVL_PSL
  `include "./psl11/assert_always_logic.v"
`endif // OVL_PSL

`endmodule
```

Figure 2. New OVL assert_always Implementation

For this checker, the interface is exactly the same between the different implementations: Verilog, SystemVerilog, and PSL. In fact, the only difference between the flavors of this checker is in the implementation details. For the Verilog version

(enabled by the OVL_VERILOG macro), procedural Verilog source code is included as part of the checker. For the SystemVerilog version (enabled by the OVL_SVA macro), source code is included as part of the checker. Similarly, the PSL version (enabled by the OVL_PSL macro) includes source code that permits binding a PSL vunit, which contains the required PSL assertion constructs, to a Verilog placeholder module.

Note the use of the `module and `endmodule macros (which are defined in the std_ovl_define.h file). The library was implemented to allow the macros to be redefined as SystemVerilog interfaces (versus the default module definition), thus providing extended power to SystemVerilog users.

4.2 Timing Diagrams

Version 1.5 of the OVL includes tutorial material that illustrates each OVL assertion with one or more *timing diagrams*. The purpose of these timing diagrams is twofold:

1. Provide a user with an illustration of what each OVL assertion is checking.
2. Provide a language-neutral visualization of the semantics of the OVL assertion, which can be thought of as its specification.

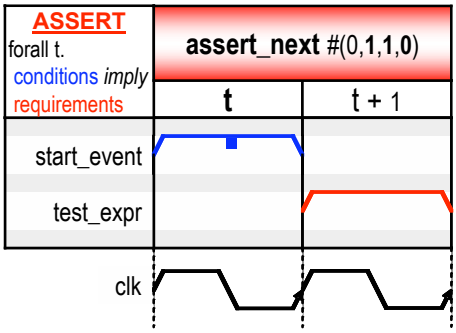

Figure 3 - Timing Diagram for assert_next

Each timing diagram consists of a number of conditions (shown in blue, and dotted for black-and-white printing) and a number of requirements (shown in red) that combine to form an implication:

> *Whenever all of the conditions match, all of the requirements must also hold.*

Timesteps correspond to a clock cycle, and "forall t." means that timestep t can be *any* clock cycle. Imagine sliding the timing diagram over a simulation—whenever start_event is high, it triggers the assertion and checks that test_expr is high on the following cycle. The same visualization works in formal verification, but an important difference is that to be *exhaustively proven*, the timing diagram would have to slide over all possible input sequences.

As there are no conditions at t+1, this assertion is *fully* pipelined, that is, two back-to-back start_event's require two back-to-back test_expr's (delayed by one cycle). Most timing diagrams are simple (for example, assert_always only has one timestep), but some timing diagrams involve a window of cycles, and these can also be represented by timing diagrams (sometimes with auxiliary logic if the parameters are configured to forbid pipelining).

4.3 Coverage

Since the pre-Accellera version of the OVL, each assertion has been instrumented with internal functional coverage points. These coverage points are disabled by default, but they can be enabled to report events and conditions associated with a particular assertion.

Not only are timing diagrams useful for illustrating an assertion, you can use them to illustrate and review internal coverage points. Figure 3 illustrates the assert_next assertion. An obvious coverage point is to check start_event. Clearly, if this is never high, the assertion itself will trivially pass. You can check this coverage point in different parts of the design flow:

- RTL simulations can show if start_event was ever high in a simulation or in any one of a suite of simulations.
- Formal verification can show if it is impossible for the start_event to be true (for example, a vacuous condition like A & !A) or can show a sequence to trigger this functional coverage point (which is useful if it is hard to hit by random stimulus).

In version 1.5 of the OVL, it is now possible to set the property_type parameter of an assertion to `OVL_IGNORE so that it does not check anything. Instead, it provides a functional coverage point. You can implement any sequence you want to cover using an assert_cycle_sequence OVL (illustrated below) and setting the following:

- property_type parameter to `OVL_IGNORE
- event_seq[num_cks-1] to 1'b1 (this is not checked)

In Figure 4, the sequence from t to t+1 will be checked as a functional coverage point if property_type is set to OVL_IGNORE, but event_seq[2] at t+2 will not be checked (`OVL_IGNORE removes the requirements from an OVL assertion).

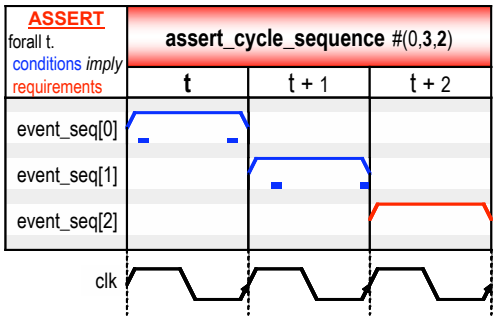

Figure 4 - Using an OVL Just for Functional Coverage

5. FUTURE OVL ROADMAP

The vision and roadmap for Accellera OVL focuses on improving the efficacy of functional verification using assertion checkers and including methodology, verification engines, and standard language support.

The OVL committee is currently working on release 2.0, which is planned for release in the summer of 2006. It will include the following key features.

- PSL and VHDL checkers
- Improved control of checker reporting
- Individual control of checker activation/deactivation

- Synthesizable features, such as output port for error identification (to be used by emulator and FPGA prototyping)
- Enhancement in all checkers for X-handling

6. A UNIFIED ABV FRAMEWORK

The OVL provides a unified ABV framework because of its compatibility across different tools and design/verification languages. Even within the same language, OVL can be applied to different methodologies [7,10].

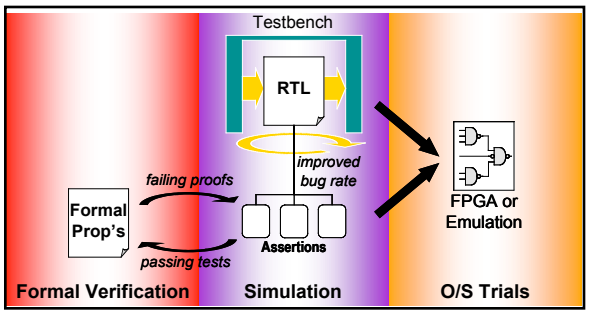

Figure 5 – OVL Assertions Used throughout Verification

Figure 5 illustrates how a number of formal and dynamic verification methods can check OVL assertions throughout the design flow. It is important to distinguish between the role of OVL assertions and verification:

- OVL Assertions: Improve *Observability* (find more bugs and their causes)
- OVL Assertions: Also improve *Controllability* (via their built-in functional coverage points)
- Dynamic Verification: Checks the design using a wide range of techniques (simulation, emulation, and FPGA prototyping that requires a synthesizable version of OVL)
- Functional Coverage: Improves *Controllability* of Dynamic Verification
- Formal Verification: Gives 100% *Controllability* (all legal input sequences)

Experience demonstrates that an OVL assertion will find more bugs and will help in the debugging process itself. (They locate the root cause of a failure in the appropriate part of the design.) However, OVL assertions will only fire if the bug is stimulated. Ideally you would formally verify *all* OVL assertions to be 100% exhaustively proven, but this is currently not practical for large designs (due to the inherent complexity of formal verification). The solution is a mixed approach, where you first try to verify them all dynamically (debugging is easier using this method) and then formally verify the assertions that pass dynamically. Later in a project, there is an opportunity to reduce simulation regressions by not having to dynamically verify assertions that have been exhaustively proven.

7. OVL AS VERIFICATION IP

The OVL provides an excellent mechanism for distributing verification IP, due to its unified ABV framework (which provides compatibility across a wide variety of tools, languages, and methodologies).

One example of an IP provider is ARM, which uses OVL assertions internally across a wide range of verification methodologies [8,9]. ARM distributes the OVL in one of two ways:

- As part of the design, that is, OVL assertions embedded in RTL

  Users can enable the OVL assertions (to test their particular configuration) or use them as an integration check (to ensure the design is correctly wired).
- As a standalone piece of verification IP, for example, AMBA protocol checkers

ARM IP is widely used, so this distribution of OVL assertions gives a valuable source of verification IP to a large number of companies (including those who are new to ABV). This type of OVL distribution lets users try OVL without even writing any assertions themselves.

8. SUMMARY

In this paper we introduced the new Accellera Open Verification Library. The new OVL standard fulfills the vision of creating a vendor- and language-independent assertion library that can be leveraged across multiple verification processes. It is a completely re-architected version of the original OVL that now takes advantage of new features offered by the emerging PSL and SystemVerilog standards.

9. WHERE TO FIND OVL

To download the latest release of the library, go to:

http://www.accellera.org/activities/ovl/TermsConditions

For additional information concerning the OVL, go to:

http://www.accellera.org/activities/ovl

To access the OVL issues list under the Mantis project "*Std OVL Errata*", go to:

http://www.eda.org/mantis

10. ACKNOWLEDGMENTS

Eduard Cerny, Dmitry Korchemny, Uma Polisetti, Ramesh Sathianathan, Sundaram Subramanian

Special thanks to Mentor Graphics for its OVL SVA donation as well as valuable resources (people and machines) to develop and validate the new Accellera OVL standard.

11. REFERENCES

[1] H. Foster, C. Coelho, "Assertions Targeting A Diverse Set of Verification Tools," Proceedings of the 10-th Annual International HDL Conference, March, 2001.

[2] H. Foster, A. Krolnik, D. Lacey. Assertion-Based Design, Second Edition, Springer, 2004.

[3] IEEE Standard for Property Specification Language (PSL), IEEE Std. 1850-2005.

[4] IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language, IEEE Std. 1800-2005.

[5] Accellera Standard OVL v1.1 Library Reference Manual, 2005, www.accellera.org.

[6] L. Bening, H. Foster, Principles of Verifiable RTL Design, Second Edition, Kluwer Academic Publishers, 2001

[7] E. Seligman, R. Koganti, K. Maheswaran, R. Naqib, “Bringing Formal Property Verification Methodology To An ASIC Design”, Proceedings of DVCon, 2006.

[8] P. Hughes, M. Turpin, M. Wilder “Decomposing an ARM Architecture Specification into Assertions for Dynamic and Formal verification”, Euro DesignCon 2005

[9] M. Turpin, “Solving Verilog X-Issues by Sequentially Comparing a design with itself”, Boston SNUG 2005

[10] K. Larsen, P. Yeung, "Practical assertion-based formal verification", SoC 2005.

[11] R. Ho "Maximizing Synergies of Assertion and Coverage Points", DVCon 2005.