Improving Design and Verification Productivity with VHDL-200x

Stephen Bailey
sbailey@model.com
Model Technology, a Mentor Graphics Company

Erich Marschner
erichm@cadence.com
Cadence Design Systems

J. Bhasker
jbhasker@esilicon.com
eSilicon Corporation

Jim Lewis
Jim@SynthWorks.com
SynthWorks

Peter Ashenden
peter@ashenden.com.au
Ashenden Designs Pty Ltd

Abstract

VHDL is a critical language for RTL design and is a major component of the $200+ million RTL simulation market[1]. Many users prefer to use VHDL for RTL design as the language continues to provide desired characteristics in design safety, flexibility and maintainability[2]. While VHDL has provided significant value for digital designers since 1987, it has had only one significant language revision in 1993. It has taken many years for design state-of-practice to catch-up to and, in some cases, surpass the capabilities that have been available in VHDL for over 15 years. Last year, the VHDL Analysis and Standardization Group (VASG), which is responsible for the VHDL standard, received clear indication from the VHDL community that it was now time to look at enhancing VHDL.

In response to the user community, VASG initiated the VHDL-200x project[3]. VHDL-200x will result in at least two revisions of the VHDL standard. The first revision is planned to be completed next year (2004) and will include a C language interface (VHPI); a collection of high user value enhancements to improve designer productivity and modeling capability and potential inclusion of assertion-based verification and testbench modeling enhancements[4]. A second revision is planned to follow about2 years later. This paper focuses on the 1st revision enhancements.

[1] Gartner Dataquest 2002 EDA Forecast, CAE, RTL Simulation Market Forecast

[2] VHDL users are unconcerned with rewriting their FSM models whenever the state encoding changes or debugging race conditions as these long time Verilog shortcomings have never been an issue in VHDL.

[3] See http://www.eda.org/vhdl-200x for more information.

[4] Assertion-based verification and testbench modeling enhancements that are ready when the first 200x revision goes to ballot will be included in that ballot.

1. VHDL Programming Interface (VHPI)

A programming interface to VHDL has been defined and is in process of being integrated into the VHDL LRM. The VHPI will open the door for easy development and integration of 3rd party tools with any LRM compliant VHDL simulator and C language models.

2. Assertion-based verification

Assertion-based verification involves adding behavioral specifications to a design in order to improve verification efficiency. These specifications can define requirements on design behavior that can be checked both statically, using formal verification techniques, and dynamically, during simulation. They can define both intended design behavior, which should be demonstrated during verification, and potential error situations, which should not occur during verification. They give added visibility into the internal state of a design.

Over the past several years, Accellera[5] developed a standard language for property specification and assertion-based verification. The Property Specification Language (PSL), is based upon the language Sugar[6], developed by IBM Haifa Research Labs. PSL version 1.01[7] became an Accellera standard in May 2003. Verification tools supporting PSL have been available for more than a year with more tool support coming.

PSL has been chosen as the basis for all property specification capabilities for VHDL. PSL will be integrated as a native language capability in VHDL and exploited fully for use in assertion-based verification and verification automation (specification of constraints and

[5] http://www.accellera.org

[6] http://www.haifa.il.ibm.com/projects/verification/sugar/index.html

[7] http://www.accellera.org/pslv101.pdf

sequences for use in constrained, random stimulus generation).

3. Testbench and verification

Today many users resort to high-level verification (HLV) languages to access language features that are required for verification. However, they have clearly identified that it would be far easier to continue to use VHDL if it included the HLV features.

Based on user feedback, a number of areas have been identified that can help improve testbench writing and consequently verification. The top priorities are:

- Associative arrays
- Fork-join
- Queues
- FIFOs
- Lists
- Synchronization and handshaking (event objects)
- Request and wait for action
- Expected value detectors
- Access to coverage data for reactive TB
- Sparse arrays
- Random value generation with optional and dynamic constraints and weightings
- Random object initialization
- Random 2-state value resolution
- Loading and dumping memories

Proposals are being evaluated in the areas of associative arrays, lists, fork/join (dynamic process creation) and more.

4. Data Types and Abstraction

We are evaluating historical VHDL language enhancement research efforts, such as SUAVE[8] to determine if we can incorporate features from those efforts including:

- Object-orientation and inheritance
- Enhanced generics (generic types and subprograms)
- Dynamic process creation and destruction
- Interface modeling.

5. Modeling productivity and capabilities

The goal of the modeling and productivity group is to improve designer productivity through enhancing conciseness, simplifying common occurrences of code, and improving capture of intent. The following are examples of proposed enhancements:

- Unary reduction operators
- Array/scalar logical operators
- Hierarchical signal access
- Formatted IO and string conversion functions
- Sized bit string literals
- Conditional and selected assignments within sequential code blocks
- Case and If-else generate
- Reading driving values of output ports
- Dynamically-evaluated expressions in port maps
- Sensitivitiy list abbreviation and design intent enhancements
- Relax local static requirement for case alternatives
- Allow user-definable matching for case statements to exploit don't care matching

6. Environment

We are evaluating ways in which we can make the use of VHDL more portable and usable. Standardizing helpful utilities such as simulation stop, finish and restart routines so they are callable from anywhere in the VHDL code have been proposed. We will also consider any proposal, such as standardizing the semantics of co-simulation with other languages, within this context.

7. Performance

The working group is sensitive to all productivity considerations including tool performance. Therefore, we are evaluating ways in which the VHDL semantics can be relaxed or modified in specific contexts to improve the ability of tools to exploit more optimizations. We will be careful to not be penny-wise and pound-foolish. For example, many man-decades of valuable verification time have been wasted debugging Verilog race conditions. Therefore, we will not provide many verification automation/productivity enhancements only to subtract from those advancements by introducing poor semantics elsewhere in the revised language.

Summary

The evolution planned with VHDL-200x will allow VHDL users to exploit new features in assertion-based verification, verification automation and modeling and tool performance while they continue to reap all the current benefits of VHDL.

[8] http://www.ashenden.com.au/suave.html