SystemVerilog Interoperability Checklist

Stuart Sutherland
Sutherland HDL, Inc., Portland, Oregon
stuart@sutherland-hdl.com

Abstract

The goal of this paper is to enable design and verification engineers to begin using SystemVerilog immediately.

This paper presents a detailed list of the of the hundreds of constructs in the SystemVerilog standard. The constructs are presented in the form of a checklist. Using this checklist, engineers can evaluate their electronic design tools to determine if the tools support the specific SystemVerilog constructs to be used in a design project. Companies providing electronic design tools will also find this checklist useful as they add support for SystemVerilog to their products.

Determining a common set of SystemVerilog constructs supported in the current release of the tools to be used in a design/verification project enables engineers to benefit from many of the powerful SystemVerilog features, without having to wait for all tools to support 100% of the SystemVerilog standard. Much of SystemVerilog can be—and should be—used in current design and verification projects!

1. Introduction

SystemVerilog unifies several proven hardware design and verification languages, making the features of these languages extensions to the Verilog Hardware Description Language. SystemVerilog integrates into Verilog:

- The extended RTL modeling constructs of the SUPERLOG language
- The abstract modeling capabilities of C++
- The powerful VERA verification language
- The capabilities of the PSL assertion standard
- Capabilities in VHDL that were lacking in Verilog
- Design and verification constructs from a number of other sources.

SystemVerilog also provides functional coverage capabilities, a direct programming interface, an assertions programming interface, and a coverage programming interface.

At the Design Automation Conference (DAC) in June 2004, some 50 Electronic Design Automation (EDA) companies announced that their products either already were supporting SystemVerilog, or would soon be supporting SystemVerilog. Sutherland HDL is an early adopter of SystemVerilog. We have participated in the standardization process of SystemVerilog since its inception, and encourage the use of SystemVerilog in all current design projects.

Much to our dismay, however, at Sutherland HDL we quickly found that the marketing claims for SystemVerilog support are often exaggerated. The EDA products we have used do indeed support some of SystemVerilog, but not one of the products currently in use at our company supports every feature of SystemVerilog. And no wonder! The SystemVerilog Language Reference Manual (LRM) is over 550 pages, and this is just to document the extensions to the Verilog HDL. The combined Verilog and SystemVerilog LRMs total over 1,200 pages! [2][4]

2. The problem, and the solution

The sheer extent of the SystemVerilog standard, coupled with often exaggerated marketing claims from EDA companies, creates a problem for companies who wish to use SystemVerilog. How can a company safely begin using SystemVerilog, when different tools used in the design flow support different SystemVerilog constructs. And, how can one know for sure that a tool really supports what the provider of the tool claims?

One possible solution to this problem would be to just not use SystemVerilog. Instead, stick with the tried and true Verilog-1995 standard, which is very well supported by virtually all Verilog based design tools.

Regrettably, there are a number of companies that have taken this solution to adopting SystemVerilog. But by choosing not to use SystemVerilog at all, these companies have lost out on the many significant advantages that the SystemVerilog enhancements to Verilog have to offer. In a world where time to market can make or break a product, engineers should be utilizing for every competitive edge at their disposal. SystemVerilog can help engineers create successful designs more quickly. Just as important, SystemVerilog can help avoid many subtle design errors that can be difficult, and time consuming, to find and correct.

Avoiding SystemVerilog completely and sticking with old tools and methodologies is not the solution for when to adopt SystemVerilog.

The solution is to identify which portions of SystemVerilog can be used right now, on current projects, and begin taking advantage of those portions of SystemVerilog. Then, as the project progresses, or a new project begins, re-evaluate what portions of SystemVerilog are supported by the tools being used at that time, and begin taking advantage of those additional SystemVerilog capabilities.

This approach allows companies to benefit from SystemVerilog as quickly as possible. Early adoption of the powerful constructs in SystemVerilog can give design teams a competitive edge.

The question each design team must ask, then, is do the current releases of all the tools to be used in a design support the same SystemVerilog features? If not, what is the common subset of features currently supported that can be safely used in current design and verification projects?

These important questions need to be answered with a high degree of detail and accuracy. For example, it is not enough to just ask the simple question, "*Do all the tools I use support the SystemVerilog structures?*". No doubt eager sales representatives from every EDA tool supplier will say "*Yes, we support structures*".

To really determine if a specific SystemVerilog construct, such as structures, can be used in current projects, one needs to probe deeper and determine details such as: Are both packed and unpacked structures supported? Are the data types to be used within structures supported? Are typedefs of structures supported? Is using structures with net data types supported? Do the tools being used support passing structures to tasks and functions? Do they support passing structures through module ports? Is importing structures from a package supported?

This same level of detail must be examined for every SystemVerilog construct that is to be used in a project, and must be answered for every EDA tool to be used on that project.

This paper does not answer these questions! (You didn't think it was going to be that easy, did you?)

Instead, this paper is a key enabler to make informed decisions. The purpose of this paper is to enable you, as a design engineer, verification engineer, or project manager, to accurately determine the answers on how well your tools—both commercial tools and internal tools—support specific SystemVerilog constructs.

To enable a detailed analysis of how well tools support SystemVerilog, this paper presents a ***SystemVerilog Interoperability Checklist***. This is a detailed list of *every* SystemVerilog enhancement to Verilog. The Checklist breaks down each of these enhancements to a fine level of detail; the detail needed to determine the interoperability of each SystemVerilog construct across all of the EDA tools to be used on a project.

This *SystemVerilog Interoperability Checklist* is aimed primarily at the end users of EDA tools: design engineers and verification engineers. The Checklist is provided as an aid in determining which features of SystemVerilog can be used in current or planned design projects. The Checklist is also useful for selecting the appropriate EDA tools for current and future design projects. It is the author's opinion that by making educated choices on tools and language feature sets, a large portion of the powerful SystemVerilog enhancements to Verilog can be used right now.

Though this paper is targeted towards end users of SystemVerilog, EDA vendors who are adding SystemVerilog support to their software tools will also find the extensive and detailed Checklist of the features in SystemVerilog to be a valuable asset.

3. Checklist organization

The scores of enhancements that SystemVerilog adds to Verilog address two major aspects of hardware design: modeling the design functionality, and verifying the design functionality. These two major divisions can be further subdivided into specific aspects of modeling and verification.

The Checklist presented in this paper divides the SystemVerilog enhancements to Verilog into the following eight categories:

> NOTE: The categories used in this Checklist in no way imply any subsets to the SystemVerilog standard, and should not be construed as such. The categories are merely for convenience in organizing a very large spread sheet.

1) **Productivity constructs** — these are enhancements to the Verilog HDL that do not add any significant new capability, but which make Verilog easier to use. Ease of use, in and of itself, is significant. Making Verilog easier, enables writing more lines of code, while lowering the risk of making errors as the code is created.
2) **Data encapsulation constructs** — these are constructs that enable working with large blocks of data as a whole. These constructs are important for modeling very large designs that contain tens of thousands, possibly even millions, of internal signals.
3) **RTL modeling constructs** — these constructs make it possible to write models that both simulate and synthesize more accurately. All constructs listed in this category should be supported by synthesis tools.
4) **Abstract modeling constructs** — this category contains constructs that are useful for modeling designs at higher levels of abstraction than synthesizable RTL models. The constructs listed in this category may, or may not, be supported by synthesis compilers.
5) **Assertion constructs** — this category contains the constructs that make up SystemVerilog's advanced assertion-based verification capabilities.
6) **Static verification constructs** — the constructs in this category are primarily intended to enhance the verification capabilities of Verilog. The constructs are static in nature, meaning they exist from the beginning of simulation and throughout the entire simulation.
7) **Dynamic verification constructs** — these are constructs that can create objects on the fly, while simulation is running. They enable using modern, object-oriented, verification methodologies.
8) **API constructs** — these constructs include enhancements to the Verilog VPI interface, a new Direct Programming Interface, an assertion interface, and a coverage interface.

4. Checklist format

The *SystemVerilog Interoperability Checklist* is in a spreadsheet format. The columns of the Checklist are straight forward, and should require little explanation. They are:

a) A brief description of a specific SystemVerilog construct.
b) The clause in which this construct is described in the P1800-2005 SystemVerilog Language Reference Manual[1].
c) A column that can be used to indicate if this construct is important in a specific design project.
d) Multiple columns where that can be used to indicate if a specific EDA tool supports this construct.

5. The Checklist

The *SystemVerilog Interoperability Checklist* contains more than a thousand lines. This far exceeds the length allowable for papers at this conference. In addition, providing the Checklist in the form of a PDF file would limit the usefulness of the Checklist. For these reasons, this paper only includes a sample of the complete Checklist. This sample is listed at the end of this paper.

This sample of the *SystemVerilog Interoperability Checklist* serves to illustrate the Checklist contents and organization. The sample is large enough for readers of the paper to determine if the Checklist will be of use in their design projects.

The actual *SystemVerilog Interoperability Checklist* is in the form of a Microsoft Excel spreadsheet. The spreadsheet is freely available for download from the author's web site. The URL for downloading the Checklist is:

http://www.sutherland-hdl.com/sv_interop_checklist.zip

6. Conclusion

SystemVerilog provides hundreds of extensions to the Verilog standard. These extensions to Verilog address two major shortcomings in the HDLs of the 1990s. First is the ability to efficiently and accurately model ever increasing design sizes. Second is the ability to verify these extremely large, complex designs.

The adoption of SystemVerilog in current and future design projects will give companies a competitive edge. The advanced capabilities of SystemVerilog can significantly shorten design and verification cycles, and can help ensure that large, complex designs will work as intended.

EDA vendors are rapidly updating their Verilog products to support the SystemVerilog extensions. However, due to the large number of constructs in SystemVerilog, and the complexity of many of these constructs, it would be unrealistic to expect any EDA company to implement all of SystemVerilog in one release of a design tool.

The incremental adoption of SystemVerilog in software tools places a burden on engineers to carefully evaluate all tools to be used in a project, and select which SystemVerilog features can be used for that project.

The S*ystemVerilog Interoperability Checklist* presented in this paper enables design engineers, verification engineers, and engineering managers to evaluate the tools to be used in a design project. By identifying a common set of SystemVerilog constructs that are supported in the current releases of EDA tools, engineers can take advantage of SystemVerilog much earlier than waiting for 100% support in every tool.

Design and verification engineers can, and should, be using as much of SystemVerilog as possible on current design projects!

7. References

[1] *"SystemVerilog 3.1a: Accellera's Extensions to Verilog"*, Accellera, Napa, California, 2004.

[2] *"P1800-2005/D3, proposed IEEE standard for SystemVerilog"* (draft 3, pre-ballot review draft), IEEE, Pascataway, New Jersey.

[3] *"IEEE Std. 1364-2001 standard for the Verilog Hardware Description Language"* (draft 5, pre-ballot review draft), IEEE, Pascataway, New Jersey, 2001.

[4] *"P1364-2005/D5, proposed standard for the Verilog Hardware Description Language"*, IEEE, Pascataway, New Jersey.

[5] *"SystemVerilog for Design Engineers"*, Stuart Sutherland, Simon Davidmann, and Peter Flake, Kluwer Academic Publishers, Boston, Massachusetts, 2004. ISBN: 1-4020-7530-8.

[6] *"Verilog 2001: A Guide to the new Verilog Standard"*, Stuart Sutherland, Kluwer Academic Publishers, Boston, Massachusetts, 2001. ISBN: 0-7923-7568-8.

8. About the author

Mr. Stuart Sutherland is a member of the IEEE 1800 SystemVerilog standards committee for SystemVerilog, and is the technical editor of the P1800 SystemVerilog Reference Manual. He has been involved with the definition of SystemVerilog since its inception, and was founding member of the Accellera standards committee that created SystemVerilog. He is also a member of the IEEE 1364 Verilog Standards Group. Mr. Sutherland is an independent Verilog consultant, specializing in providing expert training on Verilog, SystemVerilog and the Verilog PLI. He can be contacted by e-mail at *stuart@sutherland-hdl.com*.

Appendix
Sample of the SystemVerilog Interoperability Checklist

This appendix contains a sample of the full *SystemVerilog Interoperability Checklist*. The complete Checklist is in Microsoft Excel spreadsheet format. The Checklist can be downloaded for free from the author's web site, at **http://www.sutherland-hdl.com/sv_interop_checklist.zip**

SystemVerilog RTL Enhancements

SystemVerilog Construct	LRM Clause (P1800/Draft3[1])	Importance In Project	Tool A	Tool B	Tool C	Tool D
RTL procedural blocks						
`always_comb` combinational logic block						
Inferred sensitivity list	9.2					
Sensitive to signals read within block	9.2.1					
Sensitive to signals read by functions	9.2.1					
LHS variables cannot be written elsewhere	9.2					
Automatically triggered once at time 0	9.2					
Statements that block time not allowed	9.2					
`fork`...`join` groups not allowed	9.2					
Warns if block contains latch logic (optional)	9.2					
`always_latch` latched logic block						
Inferred sensitivity list	9.3					
Sensitive to signals read within block	9.2.1					
Sensitive to signals read by functions	9.2.1					
LHS variables cannot be written elsewhere	9.3					
Automatically triggered once at time 0	9.3					
Statements that block time not allowed	9.3					
`fork`...`join` groups not allowed	9.3					
Warns if block does not contain latched logic (optional)	9.3					
`always_ff` sequential logic block						
Must have one event control (sensitivity list)	9.4					
Additional event controls not allowed	9.4					
LHS variables cannot be written elsewhere	9.4					
Blocking time controls not allowed	9.4					
`fork`...`join` groups not allowed	9.4					
Warns if block does not represent sequential logic (optional)	9.4					

SystemVerilog RTL Enhancements (continued)

SystemVerilog Construct	LRM Clause (P1800/Draft3[1])	Importance In Project	Tool A	Tool B	Tool C	Tool D
Continuous assignment enhancements						
Can assign to variable types (`reg`, `logic`,...)	9.5					
Error if variable assigned value elsewhere	9.5					
Error if variable has in-line intializer	9.5					
Enhanced decision statements						
`unique if...else...if` decisions	8.4					
Warns if no conditions match	8.4					
Warns if overlapping conditions	8.4					
Optimizes like parallel_case/full_case (synthesis)	8.4					
`priority if...else...if` decisions	8.4					
Warns if no conditions match	8.4					
Optimizes like full_case (synthesis)	8.4					
`unique case/casez/casex` statements	8.4					
Warns if no conditions match	8.4					
Warns if overlapping conditions	8.4					
Optimizes like parallel_case/full_case (synthesis)	8.4					
`priority case/casez/casex` statements	8.4					
Warns if no conditions match	8.4					
Optimizes like full_case (synthesis)	8.4					
Enumerated types						
Simple declarations (e.g. `enum {stop, go} flag;`)	3.10					
Explicit base type (e.g.: `enum logic {stop, go} flag;`)	3.10					
Enum name ranges (e.g.: `enum {state[1:3]} states;`)	3.10.2					
Enumerated type assignment checking	3.10.3, 3.10.4					
Enum methods (`first`, `last`, `next`, `prev`, `num`, `name`)	3.10.4.x					
Enhanced operators						
`++`, `--` operators in expressions	7.2					
`++`, `--` operators in `for` loop step assignments	7.2, 8.5.2					
assignment operators (`+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `<<<=`, `>>=`, `>>>=`)	7.2					
(Many more RTL enhancements are listed in the full SystemVerilog Interoperability Checklist)						

[1] The full Checklist will be updated to reflect the clause number of the final IEEE 1800-2005 standard.