

Is SystemVerilog Useful for FPGA Design? (“Burn and Learn” versus “Learn and Burn”)

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

ABSTRACT

SystemVerilog has gained rapid acceptance as a powerful ASIC and custom IC design and verification language. Are FPGA designers also using SystemVerilog? Which SystemVerilog features have they found useful? This paper answers these questions based on the experiences from several companies that have recently tried using SystemVerilog for designing and verifying FPGA designs. The paper summarizes what has worked well—and what has not work well—at each of these companies.

Table of Contents

1. Introduction	2
2. The Question this Paper Does Not Answer (and the Questions It Does)	3
3. Summary of the Quantitative Survey Results	4
3.1 General information	4
3.2 FPGA design and synthesis	6
3.3 FPGA simulation and verification	7
4. Themes Evident in the Survey Comments	8
4.1 Lack of FPGA synthesis support for the SystemVerilog language	8
4.2 Lack of simulation support for the SystemVerilog language	10
4.3 Lack of support in the tool flow for the SystemVerilog language	10
4.4 Lack of awareness of SystemVerilog capabilities	10
4.5 The learning curve for SystemVerilog	11
4.6 The cost of SystemVerilog tools	12
5. Reality and Recommendations	12
5.1 Lack of awareness of SystemVerilog capabilities	12
5.2 Lack of simulation support for the SystemVerilog language	13
5.3 The learning curve for SystemVerilog	13
5.4 The cost of SystemVerilog tools	14
5.5 Lack of support in the tool flow for the SystemVerilog language	14
5.6 FPGA synthesis support for SystemVerilog	15
6. Conclusions	15
7. References and resources	16
8. About the author	17
Appendix A: Survey Results on FPGA Usage	18
Appendix B: List of Survey Comments	20

List of Figures

Figure 1: How FPGAs are verified in 2009—which is correct?	3
Figure 2: How are the FPGAs designed by survey respondents are used	4
Figure 3: HDLs used for FPGA synthesis and verification	5
Figure 4: FPGA usage correlated with the HDL used	5
Figure 5: SystemVerilog constructs used for synthesizing FPGAs	6
Figure 6: Reasons SystemVerilog was not chosen for FPGA design and synthesis	6
Figure 7: SystemVerilog constructs used for verifying FPGAs	7
Figure 8: Reasons SystemVerilog was not chosen for FPGA verification	7

1. Introduction

The purpose of this paper is to answer the questions:

- Is SystemVerilog[1] being used for FPGA design and synthesis?
- Is SystemVerilog being used for FPGA verification?
- Which synthesizable aspects of SystemVerilog are being used?
- Which verification aspects of SystemVerilog are being used?
- What types of obstacles are impeding the use of SystemVerilog (e.g. corporate culture, learning curve, tool support)?

In 2004, I presented a paper at a conference session on FPGA design, called “SystemVerilog is for Everyone”[2]. *I felt like I was presenting to corpses at a morgue!* No one in the audience showed any interest at all in SystemVerilog. At that time, the synthesizable aspects of SystemVerilog had been available for three years, and major engineering tools such as Synopsys VCS (for simulation) and DC (for synthesis) supported the design aspects of SystemVerilog. Having presented similar papers to enthusiastic ASIC designers, I was surprised that this group of FPGA designers was so indifferent about the exciting new SystemVerilog capabilities.

“OK, it was just too soon”, I reasoned, “ASIC design tools are supporting SystemVerilog, but FPGA design tools have not caught up yet.” But that was five years ago—surely the picture has changed by now! Just three months ago, I attended a conference especially for FPGA designers. Synopsys and other companies had presentations or tutorials promoting the use of SystemVerilog for FPGA design and verification. When I looked around the room, however, it still looked like the audience was comprised mostly of corpses at a morgue! When presenters would ask the attendees if they were using SystemVerilog yet, few, if any, hands were raised.

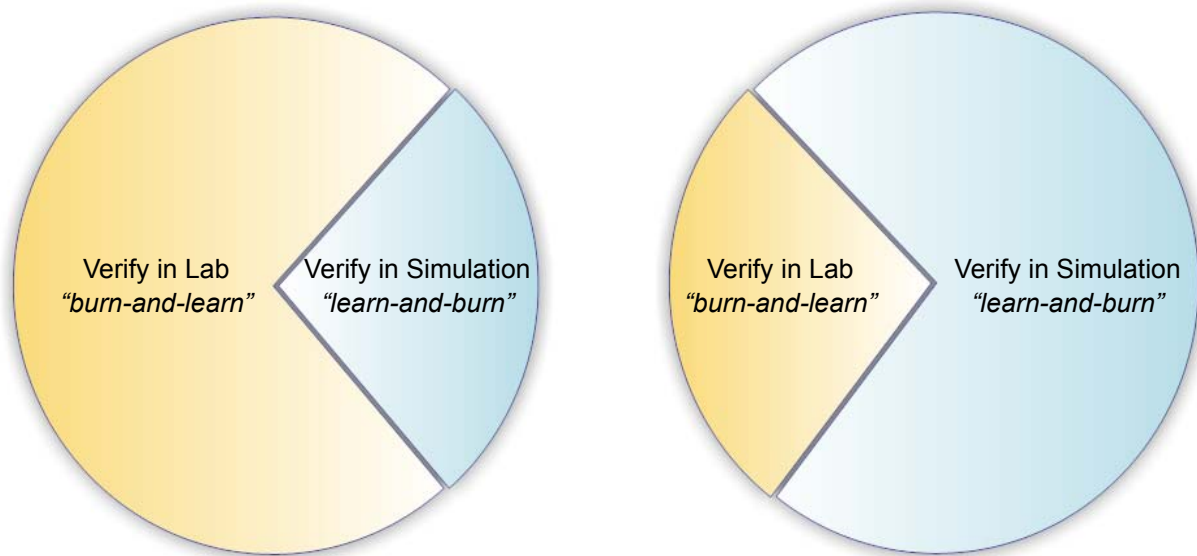
I wondered what the obstacles might be that were impeding the adoption of SystemVerilog for FPGA design and verification. Five years ago one could argue (whether right or wrong) that FPGAs were simple enough that they could be designed without using ASIC techniques such as simulation and RTL synthesis. Gates could be placed and connected using schematic capture tools, and verification could be done by programming the FPGA and testing the device in a hardware lab. Most FPGAs were simple enough to be designed using a “burn and learn” design and verification paradigm.

But that is no longer the case! FPGAs can be every bit as complex as an ASIC, with hundreds of thousands of gates, embedded processors and IP, complex busses such as PCI express and DDR-3, FIFOs, caches, and everything else that requires advanced design and verification techniques. So why isn't SystemVerilog popular with FPGA designers like it is with ASIC and custom IC designers?

2. The Question this Paper Does Not Answer (and the Questions It Does)

Which of the following pie charts is correct?

Figure 1. How FPGAs are verified in 2009—which is correct?



This paper does not answer this question! The purpose of this paper is to understand if SystemVerilog is useful for the design and verification of FPGAs. It is not necessary to know what percentage of engineers verify FPGAs using simulation (“learn and burn”) in order to accomplish this objective. To understand the usage of SystemVerilog for FPGA design and verification, this paper is focused on the experiences of engineers who are using simulation, regardless of whether these engineers are in the minority or the majority.

This paper examines experiences at more than 35 companies who either have used—or have decided not to use—SystemVerilog for designing and verifying FPGAs. Information about these experiences was gathered using a two-part survey. One part of the survey allowed respondents to select from specific choices. This type of response can be analyzed quantitatively (e.g. N% of respondents stated they are using SystemVerilog to verify their FPGAs). The second part of the survey allowed users to describe their experiences with SystemVerilog in their own words. This qualitative type of response provides a more in-depth understanding of any obstacles that might be impeding the adoption of SystemVerilog. Follow-on interviews were conducted with a few survey respondents in order to gain further insights regarding their survey comments.

This was not a scientific survey with statistically accurate results, where candidates were randomly selected from a large pool representative of all FPGA designers. Instead, this survey

used what is commonly referred to as a “purposeful sample group”[3], where a specific audience is targeted for the survey because it is known that the audience will be able to supply information pertinent to the study. The results of this type of survey are valid and important. They reflect the viewpoints of engineers who actually use languages such as VHDL or Verilog for FPGA design. Their experiences are important indicators of whether SystemVerilog is useful for FPGA design and verification, and what obstacles might be impeding a more widespread adoption of SystemVerilog in this engineering arena.

Section 3 of this paper presents a summary of the quantitative questions on the survey. **Section 4** presents an analysis of comments made in response to the qualitative questions. **Appendix A** lists the full survey and the number of responses received for each question.

3. Summary of the Quantitative Survey Results

The purpose of the quantitative questions on the survey was to get a broad picture of whether SystemVerilog is being used for FPGA design and/or verification, and, if so, how it is being used. The survey was an online form. A message was mailed to an international list of companies. Sixty-six (66) engineers responded, representing at least 35 companies in at least 11 countries (23 respondents did not specify a company or country).

3.1 General information

The purpose of the general survey questions is to understand the context for the answers to more specific survey questions. **Figure 2** shows the response to a question regarding how the FPGAs designed by survey respondents are used. The survey participants could select more than one response, so there are more responses than there were survey participants.

Figure 2. How are the FPGAs designed by survey respondents are used

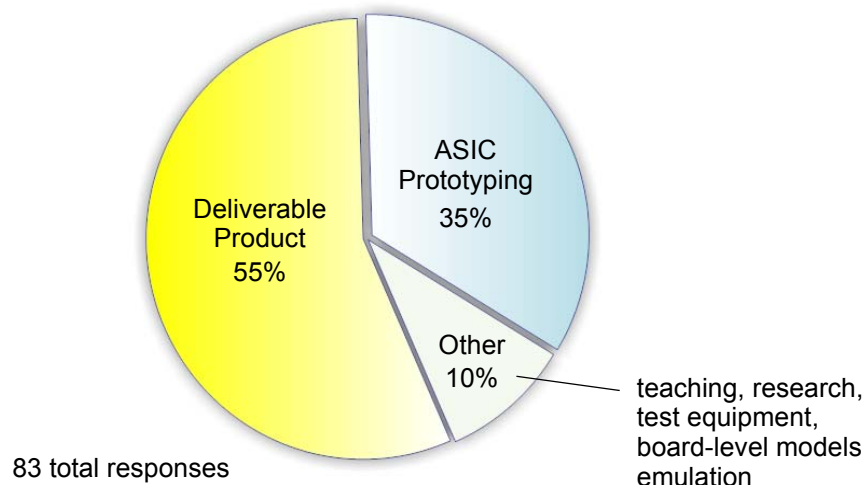
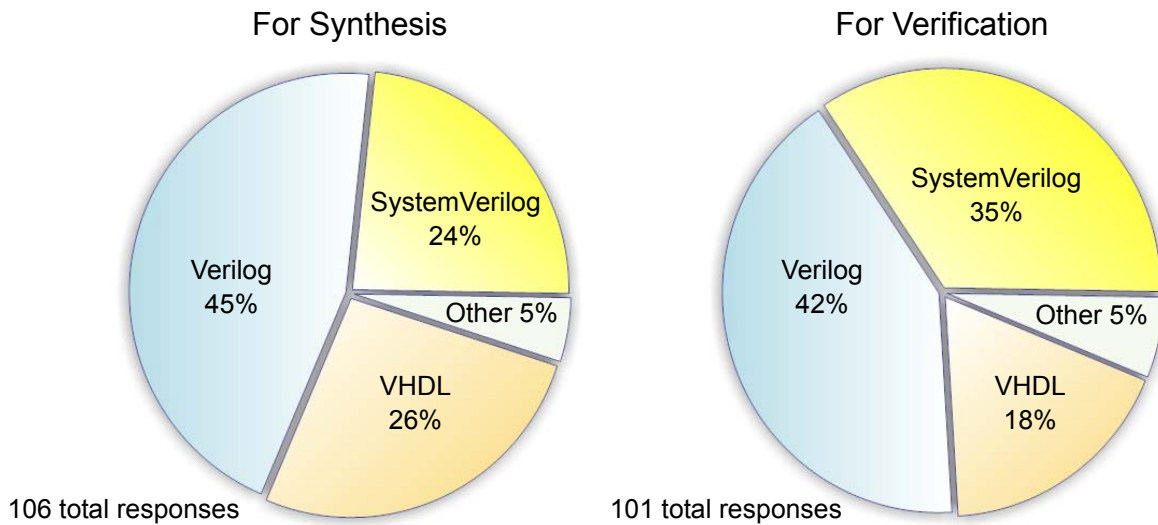


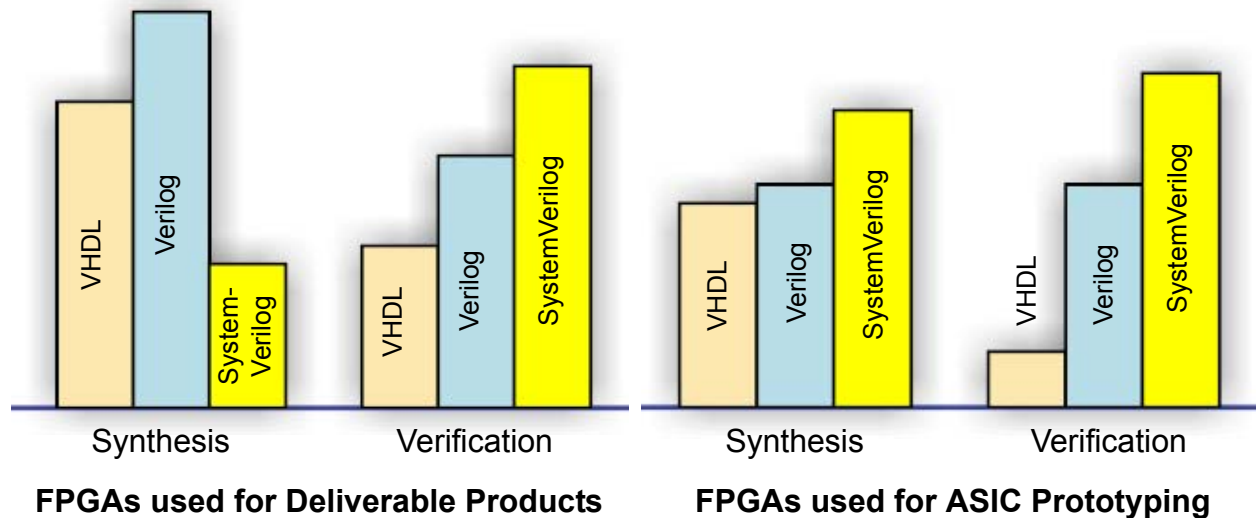
Figure 3 lists the Hardware Description Languages (HDLs) used by the survey respondents. Survey participants could select more than one HDL. Note that the study group was purposefully selected from HDL users, in order to determine if they had considered SystemVerilog for FPGA design and/or verification. The responses reflect this specific group, and might not correspond to FPGA design and verification in general.

Figure 3. HDLs used for FPGA synthesis and verification



These pie charts indicate that many engineers are using SystemVerilog for FPGA design and verification. But is SystemVerilog being used both for FPGAs intended for deliverable products as well as for ASIC prototyping? **Figure 4** correlates the data from figures 2 and 3 in order to provide an indication of which HDL is most often used for FPGAs targeted for deliverable products, and which HDL is most often used for FPGAs targeted for ASIC prototyping.

Figure 4. FPGA usage correlated with the HDL used

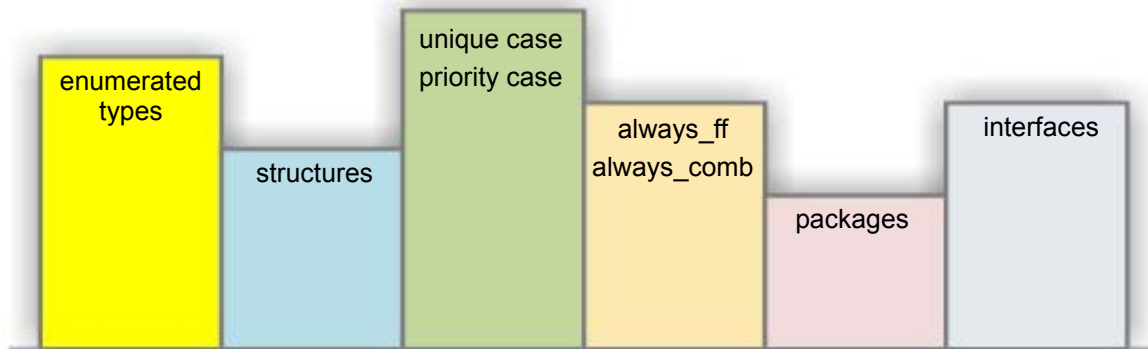


The correlation of which HDL is used with the end purpose of the FPGAs shows that SystemVerilog is used more for designing FPGAs targeted for ASIC prototyping than for FPGAs targeted for deliverable products. It is also evident that, for FPGAs, SystemVerilog is used more for verification than for design.

3.2 FPGA design and synthesis

Figure 5 applies only to survey respondents who are using SystemVerilog to design and synthesize FPGAs. The survey form allowed respondents to select if they were using specific synthesizable SystemVerilog constructs, in order to understand whether only the more basic SystemVerilog constructs (such as structures and enumerated types) are being used for FPGA design, or whether more complex constructs (such as interfaces) are also being used.

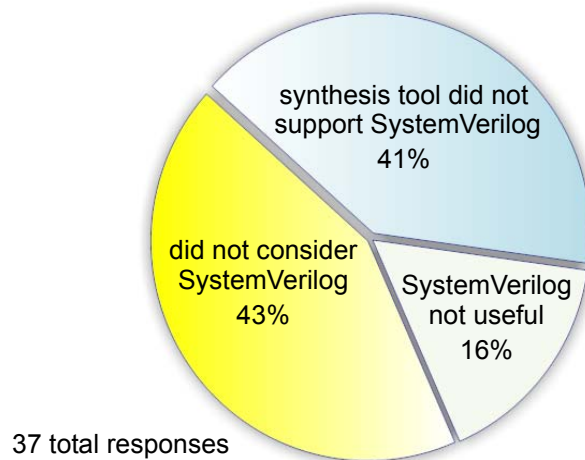
Figure 5. SystemVerilog constructs used for synthesizing FPGAs



The data above indicates that FPGA designers have used—or have tried to use—both the basic and the more complex SystemVerilog constructs. The comments survey respondents made about their experiences with using these constructs indicates that, although engineers tried to use these types of synthesizable constructs for the design and synthesis of FPGAs, there were problems. These comments are discussed in Section 4 of this paper.

Figure 6, which follows, reflects the answers given by respondents who are *not* using SystemVerilog for the design of FPGAs. The purpose of this question is to identify the primary reasons that are impeding the adoption of SystemVerilog for FPGA design and synthesis.

Figure 6. Reasons SystemVerilog was not chosen for FPGA design and synthesis

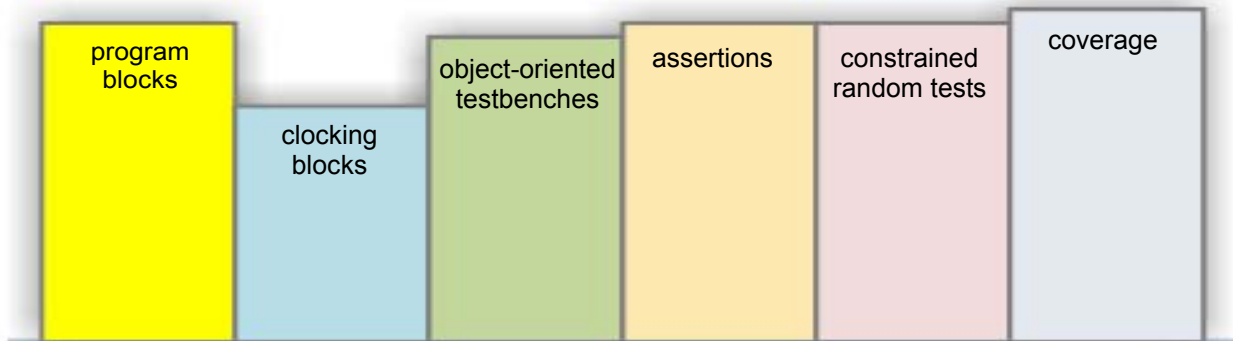


Survey participants were provided space to comment on their experiences with using SystemVerilog for synthesis. These comments are discussed in Section 4.

3.3 FPGA simulation and verification

Figure 7 applies only to survey respondents who are using SystemVerilog to verify FPGAs. The survey form allowed respondents to select from a limited list of SystemVerilog verification constructs. The list was chosen in order to gain insight into the general categories of verification constructs that are being employed. More than one response could be selected for this question.

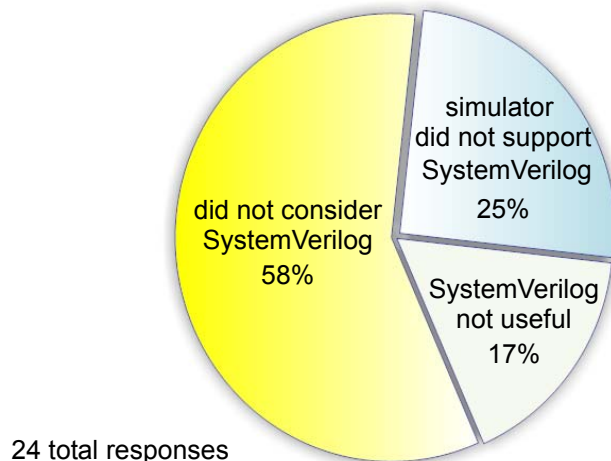
Figure 7. SystemVerilog constructs used for verifying FPGAs



It is clear that engineers who have used SystemVerilog for FPGA verification have taken advantage of all the major aspects of advanced SystemVerilog constructs.

Figure 8 reflects the answers given by respondents who are *not* using SystemVerilog to verify FPGAs. The purpose of this question is to identify a primary reason that is impeding the adoption of SystemVerilog for FPGA verification. Survey participants were only allowed to select one answer to this question.

Figure 8. Reasons SystemVerilog was not chosen for FPGA verification



Survey respondents were provided space to comment on their experiences with using SystemVerilog for verification. These comments are discussed in Section 4.

4. Themes Evident in the Survey Comments

The comments made by survey respondents along with statements gathered during follow-on interviews were analyzed to find common themes. A very clear picture emerged on barriers that have impeded the adoption of SystemVerilog for FPGA design and verification. Six recurring themes were identified. These themes, listed in the order of the most frequently mentioned to the least frequently mentioned, are:

1. Lack of synthesis support for the SystemVerilog language.
2. Lack of simulation support for the SystemVerilog language.
3. Lack of support in the tool flow for the SystemVerilog language.
4. Lack of awareness of SystemVerilog capabilities.
5. The learning curve for SystemVerilog.
6. The cost of SystemVerilog tools.

Each of these themes is explained in more detail below. Whether you agree or disagree with these themes, they reflect the reality of using SystemVerilog from the perspectives of the survey respondents. Section 5 discusses the validity some of these perspectives.

Note: This section summarizes the survey comments and interview statements. Appendix B contains a comprehensive, unedited list of all of the survey comments and interview statements that were received.

4.1 Lack of FPGA synthesis support for the SystemVerilog language

Nearly 100% of the survey respondents who have tried using SystemVerilog indicated that their synthesis compiler did not support SystemVerilog well. Synthesis compilers provided by FPGA vendors (Actel, Altera, and Xilinx were specifically named) received the worst comments. For example:

“[We need] better support in [our] vendor’s synthesis tool ([FPGA vendor name omitted] is noteworthy in not supporting it at all in their synthesis tool).”

Synthesis compilers from Electronic Design Automation (EDA) companies (such as Mentor, Synopsys, and Synplicity) received mixed comments from FPGA designers. Survey respondents rated the Synplicity (now Synopsys) Symplify-Pro synthesis compiler poorly for its support of SystemVerilog, but another EDA vendor’s synthesis compiler was rated as excellent. It should be noted, however, that many of the comments were based on experiences with these tools over the last several months. These experiences might not reflect the most current version of these tools.

For FPGA devices intended as a deliverable product, the poor support for SystemVerilog synthesis was a complete show stopper. Engineers abandoned SystemVerilog and went back to VHDL or Verilog. For engineers that were using FPGAs to prototype ASICs, the poor support for SystemVerilog synthesis was a source of frustration and aggravation that had to be worked around. One survey respondent involved in prototyping with FPGAs stated:

“I have had to go back and re-code perfectly good SystemVerilog because one tool in the chain will not support [a] construct.”

Another survey participant commented:

“I am having difficulty especially with interfaces when going from simulation (simulation works) to synthesis...where the interface was not completely supported and debugged by...[FPGA vendor name omitted].”

If the actual ASIC design uses SystemVerilog, but the FPGA prototyping tools do not support SystemVerilog, then project managers must make a difficult choice. They can force the ASIC design team to only use Verilog, or they can lose valuable time re-coding SystemVerilog code to work around the limitations of the FPGA synthesis compiler. Not using SystemVerilog at all would mean that the ASIC designers lose the advantages of using SystemVerilog for design. Rewriting SystemVerilog code to work around FPGA synthesis limitations has the risk of the FPGA not representing the same functionality as the actual ASIC.

An interview with another survey participant indicated how severe this problem is for prototyping ASICs using FPGAs. This company is in the process of designing a very large ASIC. Part of their design flow is to emulate the ASIC, using FPGAs before the ASIC is built. The emulator allows the software for the system to be developed and debugged in conjunction with designing the ASIC. Due to the complexity of the ASIC, using SystemVerilog is considered a necessity for the design and verification. It requires 90 large-capacity FPGAs to emulate the ASIC. The engineer being interviewed indicated that:

[paraphrased for conciseness] The most current versions of the two FPGA synthesis compilers being used (one from an EDA company and one from an FPGA vendor) only support about 50% of the SystemVerilog constructs supported by their ASIC synthesis compiler. In order to emulate the ASIC in FPGAs, we have to make substantial changes to the RTL code to manually re-write SystemVerilog constructs as Verilog. Worse, the re-writing must be done differently for each of the FPGA synthesis compilers, because the limited support for SystemVerilog is quite different for each compiler.

Another company prototyping an ASIC in FPGAs also complained that their FPGA synthesis compiler did not support the same SystemVerilog constructs as their ASIC synthesis compiler, even though both products were from the same EDA company. This engineering team found a way to work around the obstacle of poor FPGA synthesis support. The design flow they used in a recent project was:

“FPGA flow was DC -> GTECH -> [EDA synthesis compiler name omitted] -> [FPGA vendor name omitted].”

With this flow, synthesis was only done with an ASIC synthesis compiler (Synopsys DC). The synthesis GTECH generic gate-level netlist from ASIC synthesis was imported into the FPGA synthesis compiler to target the FPGA device. The flow worked, but had two problems. The FPGA synthesis compiler would only read in a GTECH file generated by an older version of the DC synthesis compiler, which prevented using the most current version of the ASIC synthesis tool. The second problem is that this FPGA synthesis compiler is no longer available, and so this flow cannot be used with the ASIC prototyping project the engineering team is now working on. This engineering team is now using an FPGA synthesis tool from another EDA vendor, and is

having good success with directly synthesizing the SystemVerilog models without having to work around any limitations.

4.2 Lack of simulation support for the SystemVerilog language

Approximately 20% of the survey respondents indicated that their simulator did not fully support SystemVerilog. Most of these comments were on the verification aspects of SystemVerilog. Two specific limitations that were mentioned multiple times are:

- Interfaces not fully supported
- Generate statements with parameterized models not fully supported

Some comments were also made about simulators being buggy, or only partially supporting a construct. For example:

“I have encountered a fair share of SystemVerilog related bugs in both simulation and synthesis tools during the last two years. Hopefully the state of the tools will improve.”

Another limitation for simulation that was mentioned was simulators not implementing a SystemVerilog construct in the way it is described in the SystemVerilog standard. One survey participant commented:

“I have some concern that the major vendors aren’t sticking to [the] LRM [Language Reference Manual]. Additionally, they all aren’t really trying to fill out the LRM functionality, but [instead] trying to see [what] is most used. I think this is bad. Moreover, if they aren’t going to really meet the LRM, I’d like to have them tell me what isn’t supported. This seems to be almost a trade secret.”

4.3 Lack of support in the tool flow for the SystemVerilog language

Several respondents commented that SystemVerilog was not well supported across the flow of all the tools used in their design and verification process. Synthesis compilers were the broken link in many of the tool flows (see section 4.6). One respondent specifically mentioned that their linting tool did not adequately support SystemVerilog in their tool flow. Another respondent mentioned lack of complete SystemVerilog support in their waveform/debugging tools.

“[Simulator name omitted] does not display the signals of an interface in the signal list of a module using it. Why not have them show up like the other ports? Having to add the interface instance to the watch list is a pain. The info is there, just add them.”

4.4 Lack of awareness of SystemVerilog capabilities

A significant number of the survey comments indicated that many FPGA design and verification engineers have an in-depth knowledge of SystemVerilog, and have considerable experience with using many of the advanced features of the language. However, there were also a number of comments that suggested there are a significant number of FPGA engineers who do not fully understand the capabilities of SystemVerilog, or the benefits of using SystemVerilog. For example:

“Seems like it [SystemVerilog] is just as useful as Verilog since it is a superset of Verilog....We can ‘get by’ without learning something new.”

and

“Some of our Verilog users have been reluctant to move away from Verilog 95.”

A VHDL user commented that design features of SystemVerilog were only catching up with VHDL synthesis capabilities, but not adding anything new that was of value.

“The new features [in SystemVerilog] compatible with synthesis are all mostly catch-up features already present and working in VHDL. Plus I still feel that SV inherits too much of the low level nature of Verilog and does not allow as high-level design as VHDL does....Interfaces are about the only interesting new thing, but we have been using records in VHDL for a similar purpose for a long time already with good results.”

Statements such as this indicate some FPGA designers are not familiar with how the rich set of synthesizable constructs offered by SystemVerilog enable writing more compact, efficient, bug-free hardware models.

Lack of awareness of current tool support was also evident in some of the comments. One respondent stated that they are not using SystemVerilog because tools do not support it, but then added:

“Our last assessment was done 2-1/2 years ago. We have not re-assessed since that time.”

Another respondent stated:

“I’ve found that V[erilog]-2001 is only coming into its own in the last couple years for synthesizers and simulators. I predict the cool new synthesize-able constructs in SV [SystemVerilog] won’t be universally available until 2010.”

Not understanding the benefits of using SystemVerilog impedes the adoption of using SystemVerilog. Not keeping abreast of what the latest versions of software tools can do with SystemVerilog also impedes benefitting from SystemVerilog.

4.5 The learning curve for SystemVerilog

Another re-occurring theme evident in the survey comments was the learning curve involved with adopting SystemVerilog. Specific issues mentioned included the cost of training, the time it takes for training, and the availability of training. One respondent noted that learning SystemVerilog would impact their project schedules:

“We considered SystemVerilog, but did not have adequate training and did not want a disruption.”

Another respondent stated that the lack of availability of SystemVerilog training in India was

preventing companies from using SystemVerilog.

“To adopt SystemVerilog, the main constraint we feel is lack of good training available in India.... Understanding [SystemVerilog] will [be] the fundamental starting of adopting SystemVerilog.”

4.6 The cost of SystemVerilog tools

A small number of survey respondents mentioned that the cost of simulation and synthesis tools was an obstacle for adopting SystemVerilog:

“We don’t want to pay for extra licenses, so any support [for SystemVerilog] would have to come through the simulation tools we now own. And don’t even think of trying to bump up my maintenance costs to cover that capability.”

“[SystemVerilog needs] support from a free open source simulator. I use [simulator name omitted], which presently only supports Verilog 95.”

5. Reality and Recommendations

The analysis of the survey results revealed that engineers engaged in FPGA design and verification perceive six factors that have impeded the adoption of SystemVerilog. This section of the paper looks at whether these perceptions are grounded in reality. Recommendations are made on how to remove or reduce any actual obstacles to using SystemVerilog for FPGA design and verification. The six factors are discussed in order of severity, from least severe to most severe.

5.1 Lack of awareness of SystemVerilog capabilities

Some of the comments made by survey respondents suggest that awareness of SystemVerilog capabilities is a factor impeding the adoption of SystemVerilog for FPGA design and verification. These comments indicate that many companies doing FPGA designs do not understand the benefits of using SystemVerilog. This is not surprising. Major EDA companies, including Synopsys, have focussed their marketing efforts for SystemVerilog on the advanced verification capabilities of the language, and on powerful—but complex—verification methodologies such as VVM. There has been very little marketing regarding the benefits of using the design aspects of SystemVerilog, which are considerable.

This marketing focus has two negative impacts on the adoption of SystemVerilog in the FPGA arena. First, it down plays the benefits of the design aspects of SystemVerilog. If these aspects are not promoted, engineers are likely to think that either there is no benefit to be realized, or that the design constructs in SystemVerilog are not yet supported. Second, promoting the advanced, complex SystemVerilog verification methodologies can inadvertently send the message that all aspects of SystemVerilog are difficult to learn and use.

Recommendations: To address this issue of awareness, Synopsys (and other EDA companies) should aggressively market: (a) the benefits of designing with SystemVerilog, and (b) the fact that an engineer does not need to become an Object-Oriented programming expert in order use—and benefit from—many of the SystemVerilog verification features.

5.2 Lack of simulation support for the SystemVerilog language

Many survey respondents commented on a lack of simulation support for some SystemVerilog constructs. This is more of an awareness issue than a real obstacle impeding the use of SystemVerilog for FPGA design and verification. Simulation support for SystemVerilog in current commercial simulators, such as VCS, is excellent.

Recommendations: The lack of awareness about how well simulators support SystemVerilog can be resolved with better marketing by the companies that provide SystemVerilog simulators.

5.3 The learning curve for SystemVerilog

Learning SystemVerilog is a legitimate concern, but perhaps not as severe as some FPGA engineers perceive it to be. For design and synthesis, the transition from either VHDL or Verilog to SystemVerilog is not difficult, and learning SystemVerilog should not be an obstacle. The benefits of better RTL designs should easily offset the cost and time of learning SystemVerilog. These benefits are beyond the scope of this paper, but are topic of other papers[5][6][7][8][9].

For verification, the transition from VHDL or Verilog to SystemVerilog can also be relatively simple. It is a misconception that one *must* learn Object-Oriented Programming (OOP) in order to use SystemVerilog for verification. The reality is that SystemVerilog offers substantial benefits for verification over VHDL and Verilog, without using the OOP features of the language. Second, although the OOP features of SystemVerilog enable powerful verification techniques, these features are not difficult to learn. It isn't SystemVerilog verification that is difficult, it is the complexity of the design that can make verification difficult.

Perhaps a different issue related to learning SystemVerilog is learning the RTL-based design and verification paradigm in general. Some FPGA designers are used to designing at the gate level and verifying designs in a lab, rather than in simulation. For these engineers, RTL-based design and verification in simulation is a new way of thinking about hardware. Making this paradigm shift at the same time as adopting all of the advanced verification features in SystemVerilog could be a quantum leap, rather than an incremental step. Here again, however, there is the misconception that one must learn everything about SystemVerilog before being able to use any aspect of SystemVerilog. The reality is that gate-level designers could start off with learning a much smaller subset of SystemVerilog, and gradually add to that knowledge over time.

Recommendations: Companies considering adopting SystemVerilog should start with expert-level training in specific aspects of the language required for the design project, such as synthesizable constructs and assertions. Then, as the project needs increase, take additional training on more advanced language features, such as OOP verification. SystemVerilog is a language that can be adopted in phases, rather than all at once. Expert-level training does cost money and time. It is beyond the scope of this paper to debate whether an investment in training pays for itself in productivity. Suffice it to say that, as a training provider, my company is convinced that training saves time and money in the long run.

One survey participant also had a recommendation for EDA companies:

“Popularity of the language should be improved among academics and students, from where you can expect a pool of prospective SystemVerilog programmers....High end

industry standard training should be offered to [a] select group of interested faculty members...FREE OF COST. This would be the best investment towards promotion of the language, as they are like candles which can ignite many future sources.... SystemVerilog partner companies should consider funding high quality training companies like Sutherland HDL for offering such training programs for faculty members (especially in [countries] like India, where there is literally no quality training in SV offered, but has immense manpower...in design and verification)."

5.4 The cost of SystemVerilog tools

A few survey respondents commented that the cost of SystemVerilog tools is impeding the adoption of SystemVerilog for FPGA design and verification. Even though there were only a few such comments, it is possible that this is a significant issue. Companies that design both ASICs and FPGAs have access to SystemVerilog based design and verification tools. Smaller companies that do just FPGA designs might be relying on relatively low cost tools supplied by their FPGA vendor, or that are available as open-source tools. One survey respondent noted:

"Unfortunately I can't see many teams taking [SystemVerilog for verification] up due to the cost. A simulation seat is easily more expensive than all other required FPGA design software combined and needs a well trained developer to get value out of it. But we have demonstrated increased productivity and a lower in-circuit defect rate as a result of our transition to SV for verification."

Recommendations: To improve the adoption of SystemVerilog, companies that provide lower-cost tools for VHDL and Verilog today for the design and synthesis of FPGAs should continue that tradition with lower-cost SystemVerilog simulators and synthesis compilers that support a synthesis subset of SystemVerilog. To encourage the use of SystemVerilog for verification, FPGA vendors might consider helping support the development of lower-cost open-source simulators that support many of the verification features of SystemVerilog. In the long run, however, companies involved in FPGA design that want to use SystemVerilog's advanced verification features will need to accept the fact that high-end verification will require higher-end, more expensive tools.

5.5 Lack of support in the tool flow for the SystemVerilog language

One of the more severe obstacles preventing the adoption of SystemVerilog for design and verification is that every tool used in the design process must support the same aspects of the SystemVerilog language. Synthesis is a critical link in the flow of a design through various tools, and is discussed in section 5.6, below. There are also other tools that are part of the design and verification flow that must support the same SystemVerilog constructs. If the simulator and synthesis compiler both support SystemVerilog enumerated types, for example, but a linter or waveform tools does not, then it becomes difficult, if not impossible, to use enumerated types in the design flow.

Recommendations: To eliminate this obstacle, tool vendors need to ensure consistency across all of their tools. EDA companies and FPGA vendors that provide FPGA design and verification tools need to commit the resources necessary to make all of their tools support the same SystemVerilog constructs at the same time.

5.6 FPGA synthesis support for SystemVerilog

Since nearly every FPGA designer who has tried to use SystemVerilog commented about the lack of Synthesis support, it must be concluded that this is a *real* obstacle to using SystemVerilog, and not one that is merely a perceived problem. This is contrary to the claims from the sales and marketing departments at major EDA companies and FPGA vendors, who all claim to support SystemVerilog for synthesis.

Follow-on interviews with some of the survey respondents revealed the reason for this disparity between reality and marketing claims. At the root of this obstacle for adopting SystemVerilog is that FPGA synthesis compilers do not support *the same* SystemVerilog constructs as do ASIC synthesis compilers. For companies involved in using FPGAs to prototype ASICs, it is crucial that the two types of synthesis compilers support the same synthesis subset of SystemVerilog. A common synthesis subset is also important for companies doing FPGAs for deliverable products. Very often the same FPGA design engineers must use FPGA synthesis compilers from different vendors. Trying to work with two completely different synthesis subsets for different engineering projects places a burden on the design engineers, and will ultimately impact how long it takes engineers to deliver working products.

Recommendations: Two steps need to be taken to resolve this issue. First, and foremost, Accellera (a consortium of EDA companies and engineering companies) needs to urgently define a recommended synthesis subset for SystemVerilog. This is the role—and the responsibility—of Accellera, which serves as a think-tank for defining new electronic design automation standards and promoting the adoption and usage of those standards. A paper presented at the DVCon conference in 2006 proposed a standard synthesis subset for SystemVerilog[4]. This paper, as well as other resources, could be used as a starting point for an Accellera SystemVerilog synthesis standard, and would help in resolving this critical obstacle affecting the adoption of SystemVerilog for FPGA design.

A second step that needs to be taken is the responsibility of EDA companies such as Synopsys. To promote the adoption of SystemVerilog for FPGA design and synthesis, Synopsys needs to:

- a. Publish—and publicize—the SystemVerilog synthesis subset that they have identified for DC. This document should be put in the public domain; it is for the benefit of Synopsys' customers.
- b. Ensure that DC (for ASIC synthesis) and Synplify-Pro (for FPGA synthesis) support exactly the same synthesis subset in exactly the same way. This might require frequent product releases, until 100% of the identified synthesis subset is supported in both products.

NOTE: Synopsys is singled out in this paper because this paper has been prepared for a Synopsys Users Group Conference, but this recommendation applies to other EDA vendors, as well.

6. Conclusions

The purpose of this paper has been to determine if engineers feel that SystemVerilog is useful for FPGA design and verification. If so, what aspects of SystemVerilog are most useful? If not, then what are the limitations or obstacles that are impeding the adoption of SystemVerilog in the FPGA design arena?

The survey and interviews conducted for this paper have provided grounded evidence that:

- There are many FPGA designers already using SystemVerilog!
- Those who are using SystemVerilog today have to work around limitations in software tools.
- There are six major obstacles that are impeding a more widespread adoption of SystemVerilog for the design and verification of FPGAs: limited synthesis support, limited simulation support, limited support in tool flows, lack of awareness, cost of training, and cost of tools.
- FPGA synthesis compilers are the weak link, and the biggest obstacle blocking the usage of SystemVerilog for FPGA design.

Two succinct comments from survey respondents about their experiences with using SystemVerilog to design and verify FPGAs are, perhaps, the best conclusion for this paper.

I can't understand why you wouldn't use it [SystemVerilog].

I wouldn't want to verify any other way!

The primary recommendation for FPGA engineers is to learn SystemVerilog! The days of “*burn and learn*” (verifying devices in the lab) are rapidly becoming ancient history. Today’s complex FPGA devices require a “*learn and burn*” paradigm, using simulation and advanced verification methods. The fact that there are already many engineers using SystemVerilog for FPGA design and verification is great news! It indicates that engineers are recognizing that the size, speed, and complexity of today’s FPGA devices require many of the same design and verification techniques used to design and verify complex ASICs

On the other hand, the poor support for SystemVerilog synthesis from both commercial EDA companies and from FPGA vendors is very disappointing. It suggests that, although FPGA designers recognize the benefits of SystemVerilog, FPGA synthesis vendors have been apathetic about supporting the standard. The recommendation to EDA and FPGA vendors is to unite in an effort to have Accellera define a standard synthesis subset for SystemVerilog.

The survey results reflect the experiences of the 66 survey respondents representing more than 35 companies around the world. If EDA companies such as Synopsys feel that these perspectives are not accurate, then they need to improve their marketing and customer support—and perhaps their products—in order to help their customers have positive experiences with using SystemVerilog for FPGA design and verification.

7. References and resources

- [1] “*IEEE 1800-2005 standard for the SystemVerilog Hardware Description and Verification Language*”, IEEE, Piscataway, New Jersey, 2001. ISBN 0- 7381-4811-3.
- [2] “*SystemVerilog Is For Everyone*”, by Stuart Sutherland. 2004. White paper available at http://www.sutherland-hdl.com/papers/2004-white-paper_SystemVerilog_is_for_everyone.pdf
- [3] “*Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research* (3rd edition)”, by John Creswell. Published by Pearson Education, Inc., Upper Saddle River, NJ, 2008,

ISBN 978-0136135500.

- [4] “*A Proposal for a Standard Synthesizable Subset for SystemVerilog-2005: What the IEEE Failed to Define*”, by Stuart Sutherland. presented at DVCon, March 2006. Available at http://www.sutherland-hdl.com/papers/2006-DVCon_SystemVerilog_synthesis_subset_paper.pdf
- [5] “*SystemVerilog: A Design & Synthesis Perspective*”, by Karen Pieper. Presented at the 2006 Synopsys Users Group Conference, Boston, Massachusetts.
- [6] “*Towards a Practical Design Methodology with SystemVerilog Interfaces and Modports*”, by Jonathan Bromley. Presented at the 2007 Design and Verification Conference and Exhibition (DVCon), San Jose, California.
- [7] “*Seamless Refinement from Transaction Level to RTL using SystemVerilog Interfaces*”, by Jonathan Bromley. Presented at the 2008 Synopsys Users Group Conference, San Jose, California.
- [8] “*Building Polymorphic Modules with Synthesizable SystemVerilog Constructs*”. by B. Hook. Presented at the 2008 Synopsys Users Group Conference, San Jose, California.
- [9] “*SystemVerilog for Design, second edition*”, by Sutherland, Davidmann, and Flake. published by Springer, Boston, Massachusetts, 2006. ISBN: 978-0-387-33399-1.

8. About the author

Stuart Sutherland is a member of the IEEE 1800 working group that oversees both the Verilog and SystemVerilog standards. He has been involved with the definition of the Verilog standard since its inception in 1993, and the SystemVerilog standard since work began in 2001. In addition, Stuart is the technical editor of the official IEEE Verilog and SystemVerilog Language Reference Manuals (LRMs). Stuart is an independent Verilog consultant, specializing in providing comprehensive expert training on the Verilog HDL, SystemVerilog and PLI. Stuart is a co-author of the book “*SystemVerilog for Design*” and is the author of “*The Verilog PLI Handbook*”. He has also authored several technical papers on Verilog and SystemVerilog, which are available at www.sutherland-hdl.com/papers-by-sutherland.php. You can contact Stuart at stuart@sutherland-hdl.com.

Appendix A — Survey Results on FPGA Usage

A total of 66 responses were received, representing more than 35 companies from at least 11 countries.

FPGA Usage

1. How are the FPGAs designed by your engineering group used?
 - Part of a deliverable product **46** respondents checked this item.
 - To prototype ASICs **29** respondents checked this item.
 - Other (please specify) **8** respondents checked this item. The usages specified were teaching, research, emulation, test equipment, and board-level modules.

Designing Functionality

2. Which Hardware Design Language (HDL) do you use to model and synthesize your FPGAs? (check all that apply)
 - VHDL **28** respondents checked this item.
 - Verilog **48** respondents checked this item.
 - SystemVerilog **25** respondents checked this item.
 - Schematic Capture **2** respondents checked this item.
 - Other (please specify) **3** respondents checked this item (no other HDLs were specified).
3. If SystemVerilog is used, please check which constructs you use:
 - enumerated types **19** respondents checked this item.
 - always_comb, always_ff **22** respondents checked this item.
 - unique case, priority case **16** respondents checked this item.
 - structures **13** respondents checked this item.
 - interfaces **16** respondents checked this item.
 - packages **10** respondents checked this item.
4. If SystemVerilog is not used, please indicate the main reason why not:
 - We considered SystemVerilog, but our software tools did not adequately support it for modeling and synthesis. **15** respondents checked this item.
 - We considered SystemVerilog, but felt it was not useful in our projects. **6** respondents checked this item.
 - We have not considered using SystemVerilog for design and synthesis. **16** respondents checked this item.
5. In your opinion, what is needed to make SystemVerilog more useful for modeling and synthesizing FPGAs?
(see Appendix B for all comments)

Verifying Functionality

6. Which Hardware Design Language (HDL) do you use to verify your FPGA functions correctly? (check all that apply)
 - VHDL **18** respondents checked this item.
 - Verilog **42** respondents checked this item.
 - SystemVerilog **35** respondents checked this item.
 - Other (please specify) **1** respondent checked this item.(SystemC)
7. If SystemVerilog is used, please check which constructs you use: (check all that apply)
 - program blocks **23** respondents checked this item.
 - clocking blocks **17** respondents checked this item.
 - object-oriented testbenches **22** respondents checked this item.
 - functional coverage **23** respondents checked this item.
 - constrained random tests **23** respondents checked this item.
 - assertions **24** respondents checked this item.
 - VMM methodology **9** respondents checked this item.
 - OVM methodology **8** respondents checked this item.
 - Other methodology **8** respondents checked this item.
8. If SystemVerilog is not used, please indicate the main reason why not:
 - We considered using SystemVerilog, but our software tools did not adequately support it for verification. **6** respondents checked this item.
 - We considered SystemVerilog, but felt it was not useful in our projects. **6** respondents checked this item.
 - We have not considered using SystemVerilog for verification. **14** respondents checked this item.
9. In your opinion, what is needed to make SystemVerilog more useful for verifying FPGAs? (see Appendix B for all comments)
10. Has your engineering team used SystemVerilog in ASIC prototyping flows?
 - Yes **14** respondents checked this item.
 - No **47** respondents checked this item.
11. In your opinion, what is needed to make SystemVerilog more useful for using FPGAs to prototype ASICs? (see Appendix B for all comments)
12. Do you have any additional comments on using SystemVerilog for FPGA design and verification? (see Appendix B for all comments)

Appendix B — List of Survey Comments

This Appendix contains the raw, unedited comments from survey respondents. The comments were used to extract the themes discussed in the paper in Section 4.

Any claims about products and languages are the opinions of the survey respondent, and might not be true. Grammar and spelling are exactly as submitted by the respondents, and were not corrected. Company names and product names have been replaced with “[name omitted]” or a similar comment in square brackets. The names and affiliations of the respondents are omitted to protect privacy.

What is needed to make SystemVerilog more useful for modeling and synthesizing FPGAs?

Seems like it is just as useful as Verilog since it is a superset of Verilog. We don't use any of the SystemVerilog constructs because we are just not comfortable with it. We can “get by” without learning something new.

Hierarchical interfaces - the language supports, but not enough tools do.

It is a hard sell for a VHDL house to use SV for synthesis right now. The new features compatible with synthesis are all mostly catch-up features already present and working in VHDL. Plus I still feel that SV inherits too much of the low level nature of Verilog and does not allow as high-level design as VHDL does - ie as a simple example it is still necessary to use 3 process FSMs (and the associated overhead) in SV due to language limitations while VHDL has no such problem. Interfaces are about the only interesting new thing, but we have been using records in VHDL for a similar purpose for a long time already with good results.

More support from Synthesis Tools side, as still now all the constructs of SV are not really supported for Synthesis.

Our last assessment was done 2 1/2 years ago. We have not re-assessed since that time.

The tools I use ([EDA synthesis compiler name omitted] and [simulator name omitted]) are only just getting usable and still have poor language coverage.

Use of SV for verification and the aquisition of a quality linting tool. Our [site name omitted] location uses verilog and a quality linting tool. Here in [site name omitted], the preference is for the strongly typed security of VHDL. I am hoping that more cooperation with [site name omitted] and the use of SV for verification will lead to Verilog for synthesis and then SV. By the time we adopt SV for synthesis, vendor support won't be an issue...

Greater support for Interfaces. Their even very useful for the top level IO and internal interfaces. [FPGA vendor name omitted] synthesis supports their use, [EDA vendor name omitted] does not. Requiring that it be fully connected on both ends is painful. Why not treat treat unconnected ends like regular IO that is unconnected. Arrays with

both packed and unpacked dimensions can be problematic. An easy way to convert packed to unpacked and vice versa would be nice. I think bit streaming may do this, but is not yet supported by synthesis. Nor, have I found good documentation on it.

Better support for the advanced language features in the synthesis tools

Stronger General Momentum, Better support from Asic tools and flows, Educated users

Tool support (synthesis, simulation, vpd viewers)

A simple guideline book will be good. The book should be simple enough for the SV beginner.

To adopt Sytem Verilog, the main constraint we feel is lack of good training available in INDIA. Tool Support is important of course. Not able to perfectly make out where System Verilog have the real edge in performing the tasks which Verilog Cannot do, understanding which will the fundamental starting of adopting System Verilog

We plan to incorporate SystemVerilog into our verification curriculum, but we are waiting for wider SystemVerilog synthesis support before using it to teach FPGA design.

Widespread support in all synthesis tools, both ASIC and FPGA ([FPGA vendor name omitted] / [FPGA vendor name omitted]).

I didn't include that we use system verilog, but if we did, the items marked would be the most used I believe.

Haven't used it yet but from know it I can see no reason. I just don't have the tools.

Better tools support.

Better support in vendor's synthesis tool ([FPGA vendor name omitted] is noteworthy in not supporting it at all in their synthesis tool).

From our perspective, we dont want to pay for extra licenses, so any support would have to come through the simulation tools we now own. And don't even think of trying to bump up my maintenance costs to cover that capability.

Tool support. I have had to go back and recode perfectly good system verilog because one tool in the chain will not support the construct.

Verilog for RTL, some SV for verification. I've found that V-2001 is only coming into its own in the last couple years for synthesizers and simulators. I predict the cool new synthesize-able constructs in SV won't be universally available until 2010.

If all synthesis and FPGA tools support System Verilog it should be fine.

Support for synthesizing SV constructs from FPGA tool vendors. SV is partially

supported by these vendors.

Language support from synthesis tool vendors.

I am having difficulty especially with interfaces when going from simulation (simulation works) to synthesis ([simulator name omitted]) where the interface was not completely supported and debugged by the synthesis vendor. I was using interfaces to port test signals up to the top level design to use with [FPGA vendor tool name omitted] for debug because I thought this would be a much cleaner approach to bring out test signals. This did not work out for me because of bugs with [EDA synthesis compiler name omitted] and I have not had time to debug this problem with [EDA company name omitted].

on the Synthesis/RTL design side - Interfaces, always_comb, ff, latch, enumeration, etc.

In the question above, another option might be pertinent: “We considered SystemVerilog, but did not have adequate training and did not want a disruption.”

I am not sure why top-level/management is not considering it as an option. I found it interesting. But languages like Esterel, BlueSpec have very high level of abstraction which is some kind of advantage which they might be considering. For example in case of Esterel which is a language tightly coupled with control intensive IPs and has strong formal tool in its suit to ensure design correctness. But the major disadvantage of all the above languages is the inability to retain signal names as in the original source code. This makes debugging hell. But I am really not sure why it is used putting our verification guys in deep trouble debugging failures in design.

To be honest, we have not tried synthesizing SV code with FPGA tools at this point - we have not taken the time to evaluate this process. It was just easier to use vanilla V2K.

My experience is that both an internal champion and management support are required for adopting SystemVerilog for design. The usual downsides limit adoption - learning new language constructs and tools which has immediate schedule impact. Reasonably priced FPGA tools would also be a big help. [EDA vendor name omitted] wants ~40 - 60k for [simulator name omitted], even if you only want assertions. An assertion only SystemVerilog license would be helpful. When [FPGA vendor name omitted] supports SystemVerilog, then large numbers of FPGA designers will adopt the flow - not before. They do not currently support SystemVerilog. To their credit, [EDA vendor name omitted]'s [synthesis compiler name omitted] seems to support a bunch of SystemVerilog constructs

In the Russia it needs more tutorials and books

What is needed to make SystemVerilog more useful for verifying FPGAs?

We are still trying to use the full extent of the language. It is huge and very powerful.

Again, our last assessment was done 2 1/2 years ago. We have not re-assessed since that time.

OVM v2.0 support

Our next project. We fully intend to use SV for any future major verification efforts.

The current [Simulator name omitted] software now does an extremely good job of linking VHDL and SV together, even when using high level constructs at the interface boundary (records, etc). Nothing specific too our FPGA flow has really got in the way. The main problem has been simulation software bugyness. Unfortunately I can't see many teams taking this up due to the cost. A simulation seat is easily more expensive than all other required FPGA design software combined and needs a well trained developer to get value out of it. But we have demonstrated increased productivity and a lower in-circuit defect rate as a result of our transition to SV for verification.

[Simulator name omitted] does not display the signals of an interface in the signal list of a module using it. Why not have them show up like the other ports. Having to add the interface instance to the watch list is a pain. The info is there, just add them. Also, we have the modport extension on the module instantiation.

cheap oEM version of [simulator name omitted] supporting SV, instead of [simulator name omitted]

Consensus among users and tool suppliers. The language is vague in so many aspects that all of the constructs checked above can not be used in the same suite.

A simple guideline book will be good. The book should be simple enough for the SV beginner.

Tool Support is important of course. Not able to perfectly make out where System Verilog have the real edge in performing the tasks which Verilog Cannot do, understanding which will the fundamental starting of adopting System Verilog

Support from a free open source simulator. I use [simulator name omitted], which presently only supports Verilog 95.

We verify our IP cores in a full system verilog environment. A simple verilog testbench is used to verify the functionality and interconnects of the FPGA port (core plus peripherals plus wrapper)

I would love to use system verilog, we just need better tool support for it.

Better support of parameterization and generates. FPGA code is heavily parameterized, where the specific configuration is chosen as needed. We've seen problems with parameterized classes and interfaces, and the layer where a "string" parameter (testbench) gets converted to a "register" parameter (dut). We also have problems measuring and merging coverage due to the DUT code changing based on parameterized

configuration.

The real bang for the buck is on DV/Modeling side. classes, structures, packages, bind, SVA, Functional Coverage, Randomization. Also, it allows us to use AVM/OVM which is a big plus.

We may be moving toward SystemVerilog testbenches at the request of a major customer.

For our company, more internal training on the constructs

Lots of older FPGA designers and embedded programmers do not know OOP. Seems like some knowledge of OOP would be helpful for the testbench constructs in SV. Wider knowledge of the benefits of assertions could also convince both designers and management to adopt assertions

In the Russia it need more tutorials and books

What is needed to make SystemVerilog more useful for using FPGAs to prototype ASICs

Synthesis Tools to allow SV constructs as synthesizable

Absolute confidence that the tool flow supports all necessary features.

Have not used it to prototype an ASIC yet.

Synthesis tools support, which is extremely poor at the moment.

Nothing. We plan to use it in the near future.

We only use FPGAs for ASIC prototyping. We just need the tools to better support system verilog to use it.

Better support from tools

auto-translation/optimization of macro blocks (such as memory arrays) to FPGA memory.

Support all of the synthesizable SV constructs! [FPGA vendor tool flow] doesn't support SV. [FPGA vendor tool flow] doesn't support it.

Additional comments on using SystemVerilog for FPGA design and verification

I can't understand why you wouldn't use it...

You need to be able to write code and be sure that reasonable RTL, that is part of the language that should synthesize, will synthesize. VHDL has had good language coverage

for many years and system verilog still has some way to go.

Was a VHDL coder for ASIC development. Now doing FPGAs with SV and would hate to use VHDL again.

As someone who has worked in the broader industry for many years and knows many computer programming languages of various kinds, I feel that SV is one of the most horrible languages I have seen. In so many ways its design ignores current thinking on language design and, cynically, appears to be a brutalized combination of E and Vera without enough effort to bring them to a common generality. To me, it simultaneously represents the 'best' the EDA industry can do and why EDA is in such dire straights. And yet, despite all that, it is definitely the best choice for verification today.

More thorough coverage of the language in synthesis.

- State machine synthesis ignores case defaults. Do turn on synthesis feature to reset state machine in uncovered states, but why not utilize case default? Sometimes the reset state is not what one wants the default to be.
- IP from FPGA vendors rarely use Verilog 2001, let alone SystemVerilog(IEEE 1800). This makes their code much harder to follow and prevent applying SV on an entire design. Must apply SV to our files and not IP files. [Simulator name omitted] now treat parameters inside a module like a localparam, can't be defparamed, only params on module can be. IP models never use parameterized modules, still use defparam.
- Function/Task overloading.
- Allowing multiple enumerated types in a module to have the same element names. For example, can't use IDLE more than once, have to make it unique somehow.

Tools embedded into [FPGA vendor synthesis compiler name omitted], [FPGA vendor synthesis compiler name omitted] a [FPGA vendor synthesis compiler name omitted] has not (at least as I checked some time ago) support for SystemVerilog. Using [EDA vendor synthesis compiler name omitted] for Synthesis in a complex project with embedded Microblaze, and IP cores gives trouble and invites you to abandon it sooner than later. I tried [EDA vendor synthesis compiler name omitted] and it was a little more promising but kind of the same thing in the end. Keep trying once a year.

EUrope is going to resist against (System)Verilog switch.

Popularity of the language should be improved among academics and Students, from where you can expect a pool of prospective System Verilog Programmers. In a world of Open source, high end industry standard Training should be offered to select group of Interested Faculty members [I repeat INTERESTED, Please dont select people based on University Rankings] FREE Of COST. This would be the best investment towards promotion of the language, as they are like candles which can ignites many future sources. There should be high quality instructional materials and learning resources available. System Verilog Partner Companies should consider funding high quality Training companies like Sutherland-HDL, for offering such training programs for faculty members [especially in Countries like INDIA, where there is literally no quality training in SV Offered, but has immense manpower requirement, perhaps the best in the

world in Design & Verification]. [EDA vendor name omitted] may also consider opening up their Indian facility for short term hands-on workshops for Instructors.

I wouldn't want to verify any other way!

I doubt that I represent a typical user. I work on sophisticated CPU designs (multicore, superscalar, RISC-DSP, with multi-level caching) however, I do it as an individual, not as a corporation backed by wealthy investors. I have enough gut feel experience to know about how fast my design will run in various process technologies without actually doing ASIC synthesis. I use the simple, free, open source [simulator name omitted] for simulation and the free [FPGA vendor synthesis compiler name omitted] web edition for synthesis, syntax error checking, and rough area/size estimation.

Shorter FPGA design cycles mean that testbench design and core design is often happening at the same time, which generally means that we start testing the core before the testbench is even declared "complete". There are a whole suite of additional problems associated with this, but it does mean that industry standard base classes and common design practices are key. Additionally, the most recent versions of the vendors tools have fixed many of our parameterization problems, but not all- we still have some key simulator problems where the resolution of parameters within SV is sometimes happens late at object creation time rather than at elaboration time. Lastly, we've used several 1364 features such as attributes, initialization, and constant functions for some time- and we continue to see support issues with these problems. I suspect that this will continue when SV is adopted for design.

We are a blended group of VHDL and Verilog95 users. We have choosen to use 'verilog' for all future development because "that is where all of the development work is on the language". Some of our verilog users have been reluctant to move away from Verilog 95. The VHDL users want to move to SystemVerilog to gain back enumerated types and packages. We are limited because the Verilog camp uses tools that do not support any SystemVerilog.

I have encountered a fair share of SystemVerilog related bugs in both simulation and synthesis tools during the last two years. Hopefully the state of the tools will improve.

Engineers should use the struct constructs, modport/interface constructs more.

As said above for SV constructs that can be synthesized into FPGA are limited. For verification it is simply the methodology issue. ASIC prototypers are familiar with CDV, OOP etc but not FPGA designers.

I have some concern that the major vendors arent sticking to LRM. Additionally, they all arent really trying to fill out the LRM functionality, but trying to see which is most used. I think this is bad. Moreover, if they arent going to really meet the LRM, I'd like to have them tell me what isnt supported. This seems to be almost a trade secret.

online resources for general reading and lots of examples comparing other languages and

SV in terms of design and verification and showing advantages and disadvantages.

First FPGA flow was DC -> GTECH -> [EDA synthesis compiler name omitted] -> [FPGA vendor name omitted].

Second FPGA flow was RTL -> [EDA synthesis compiler name omitted] -> [FPGA vendor name omitted].

First ASIC flow was DC->Astro->PC->Astro->PC/PT->GDSII.

Second ASIC flow is DC->Astro/IC->Astro/IC->PT->GDSII.

[paraphrased from an interview] We use FPGAs to emulate a large ASIC. The ASIC is modeled using SystemVerilog, and is being verified using Vera and SystemVerilog. For emulation, the ASIC design is being partitioned across approximately 90 of the largest [FPGA vendor name omitted] FPGAs. Partitioning and synthesizing the ASIC models into 90 FPGAs has been very painful due to the fact that the ASIC is modeled in SystemVerilog. The two synthesis compilers being used are [FPGA synthesis compiler name omitted] and [EDA synthesis compiler name omitted]. Both products “barf” on about 50% of the SystemVerilog constructs used in the ASIC models, but not the same 50%. To synthesize each FPGA, we have to modify the ASIC models to work around the limitations of these tools. In order to emulate the ASIC in FPGAs, we have to make substantial changes to the RTL code to manually re-write SystemVerilog constructs as Verilog. Worse, the re-writing must be done differently for each of the FPGA synthesis compilers, because the limited support for SystemVerilog is quite different for each compiler. [EDA synthesis compiler name omitted] has not been a stable tool to work with. New versions of the product are released frequently. Each new version changes the subset of SystemVerilog that is supported. These changes are not always backward compatible with how SystemVerilog was supported in a previous version. As a standard, SystemVerilog seems “soft”. The definition of how some constructs are supposed to work has changed during this project. The stupidest thing [EDA company name omitted] has done is to kill [EDA synthesis compiler name omitted]. [EDA synthesis compiler name omitted] is good for designing a single ASIC, but has a limitation of only being able to partition a design into up to 4 FPGAs, whereas [EDA synthesis compiler name omitted] could handle partitioning into the 90 FPGAs required to emulate our ASIC. Engineers should wait at least three more years before using SystemVerilog for emulation. Perhaps by then the language and tools will have matured enough to be useful.