Panel: Future SoC Verification Methodology: UVM Evolution or Revolution?

Rolf Drechsler[†]
University of Bremen/DFKI
Germany
drechsle@informatik.uni-bremen.de

Christophe Chevallaz[§]
STMicroelectronics
Grenoble, France
christophe.chevallaz@st.com

Franco Fummi[§]
University of Verona
Italy
franco.fummi@univr.it

Alan J. Hu[§]
University of British Columbia
Vancouver, Canada
ajh@cs.ubc.ca

Ronny Morad[§]
IBM Research - Haifa
Israel
morad@il.ibm.com

Frank Schirrmeister[§]
Cadence Design Systems
San Jose, CA, USA
franks@cadence.com

Alex Goryachev[‡]
IBM Research - Haifa
Israel
gory@il.ibm.com

[†]Moderator, [‡]Organizer, [§]Panelist

***Abstract*—With increasing design complexity *System on Chip* (SoC) verification is becoming a more and more important and challenging aspect of the overall development process. The *Universal Verification Methodology* (UVM) is thereby a common solution to this problem; although it still keeps some problems unsolved. In this panel leading experts from industry (both users and vendors) and academy will discuss the future of SoC verification methodology.**

Introduction

It is a recent trend that SoCs are becoming more similar to servers. Today, many SoCs are not tied to a single application anymore, but look more like general purpose PCs and high-end servers. Smartphones are the most notable example of this development, but this can also be seen in TV chips, in-car controllers, network routers, and more. This trend is occurring in parallel to the constantly growing complexity of SoCs, which support diverse IO interfaces and devices, and have complex architectures including multiple heterogeneous cores, multi-level caches, and multiple IO bridges

Today, common practice for verification is based on the *Universal Verification Methodology* (UVM [1]), which, at the system level, relies on reusing and combining unit-level environments, followed by running real software on an SoC. This methodology leaves a large gap. In high-end systems, this gap is covered by system-level verification that focuses on HW-only system integration. This level has its own methodology, dedicated environment, set of tools, and teams. It looks at the system as a whole and is not based on reusing lower level environments.

Formal methods are a field of intensive research [2]. They have been successfully applied to verify the correctness of dedicated components on the block level or for equivalence checking, but they have not been adopted by the industry for SoC-level verification yet.

In this panel several questions around this highly relevant topic are discussed. For example:

- Is the gap in today's SoC verification methodology significant? Is it growing? Or perhaps it does not exist?
- What is the right way to close the gap, if one exists?
- Is it sufficient to extend UVM capabilities (e.g., SystemC [3], TLM [4]) or are dedicated tools and methodology needed?
- Are formal methods ready to play a significant role in SoC-level verification?

In general, the panel will address the importance of system-level verification and its unique needs—whether generators, checking, coverage, or teams.

A short summary of panelists' position statements is presented in the following section.

Panelists' Position Statements

Christophe Chevallaz, STMicroelectronics, Grenoble, France

Is verification reuse a reality?

"3D Reuse"—a reality to improve: The main challenge of the SoC verification is to change the gear in the 3D verification reuse.

Horizontal reuse ("x axis") from one SoC to derivatives requires a clear definition upfront of what needs to be generic. This will enable to focus the reuse effort and ensure the development of the dedicated infrastructure for the identified reuse points.

Vertical reuse ("y axis") from IP to SoC still has room for improvement. Integration of the Software & Hardware IPs at SoC level focuses on connectivity and it *is still one of the main tasks of the verification at SoC level*. This is mostly due to the increasing numbers of IPs to manage and their diversity. There is also work to improve the connection between IP providers and SoC integrators for better understanding of their own context (i.e.: random vs directed, visibility at SoC level).

The diagonal reuse ("z axis") at various levels of abstraction (C, RTL, FPGA,..) is mandatory due the various stakeholders involved in the verification task: architecture,

design, SW, validation teams. Mixed platforms enable the mixed teams coming from various domains to be more efficient in the project development. They have a sooner access to these platforms with elements that they are more familiar with. Smooth bridges between these platforms are a key factor to ease the *debug*.

How to fill the gap?: Relying on a standard (UVM/IP-XACT [5]) to ensure interoperability is the key. Some extra efforts have to be spent in order to define a uniform way for describing an executable programming model of an IP. This executable description done with UVM or/and IP-XACT extension has to be part of the IP delivery to ease the integration in the SoC context. The VIPs also need to be part of the delivery of the IP providers. These VIPs need to be delivered with a control mechanism compatible with the SoC context. Some solutions are based on the definition of virtual registers for the VIP control to fit for example a C centric verification approach. On top, multilayered VIP that can fit various platforms (e.g., TLM, RTL, Emulation, FPGA) need to be more deployed. Tools and methodologies also have to be developed for the definition of verification metamodel with focus on the verification platform. This metamodel used in conjunction with dynamic constraint solving mechanism can support the efficient generation of derivative verification platforms.

Use formal for what?

Formal has to be considered for hot spots at the SoC level. The main area where we foresee a bigger usage is on some key structural logic. It is applied to the connectivity checks, the distributed logic for low power management and security. Some gain of efficiency can be achieved using formal techniques on the infrastructure that is managing various levels of cache and memory subsystem. More and more interconnect fabrics embark cache coherency mechanism that requires dedicated and tricky verification. One area of improvement is in more automatic black boxing to focus the formal techniques in the complex SoC environment. On top, formal methods that are usually attached mostly on the hardware side, have to be thought to be coupled with the embedded software to enhance the debug. Another wild subject of interest is the link between formal and mixed Analog Signals.

Address system verification

The challenges of system verification are multiple. There is a deep need to create System stimuli that are representative. Today, it is not obvious to get these stimuli at the right time of the development and to have the appropriate platforms to run it. The support for different platform views with mix of abstraction levels becomes a must have; some components need faster models for efficient execution.

The analysis of the system metrics that are not only functional (e.g., performance, power, energy, temperature) requires a multi-team collaboration. First solutions are to move to an application usecases verification driven solution and to leverage on closer collaboration with the architecture and software community.

Some new tools and methodology like scenario graph based techniques ease the sharing of information between the communities. These techniques acting at a higher level of abstraction give freedom of choice for test implementation (e/SystemVerilog/C/...). They also enable to concentrate the effort in building multiple data flows at the SoC level by combining IP low level drivers.

Manage huge database

One of the big challenges is to manage the increasing amount of verification data at the SoC level. The issue is that more and more data needs to be tracked; SoC is managing more and more complex and non-functional properties, e.g., performance, power, energy, temperature.

This amount of verification data makes the decision on verification closure more complex. It also makes debugging more complex. Some initiative and homemade solutions are done to refine the verification metrics and ensure that these metrics are actionable. The goal is to use these data in a more automatic and straightforward way to ease the debug, optimize the non-regression and enable the multi-team collaboration.

For example, there is a trend to leverage on MySQL database [6] and business intelligence tool to support verification closure.

FRANCO FUMMI, UNIVERSITY OF VERONA, ITALY

The name UVM is very promising, since something which is "universal" is a dream for any designer. UVM is for sure a step further from the point view of the universality of the simulation tool, however, it is not for sure a universal approach with respect to the design and verification language. That is, the methodology is too language centered, while an ideal universal verification methodology should be language independent: verification is not a matter of syntax, but of semantics. Thus, I will list the characteristics of a real universal verification methodology and how complex could it be to reach them, thus, how far away we are from the dream.

ALAN HU, UNIVERSITY OF BRITISH COLUMBIA, VANCOUVER, CANADA

The future of SoC verification methodology is formalization. This statement may seem controvertial, but it's a direct consequence of three facts:

Fact 1: Modern SoCs are among the most complex devices ever created by humanity, and the demand for even greater complexity continues: I expect this fact to be uncontrovertial and therefore provide no supporting evidence. Note that beyond the complexity arising from additional functionality, the end of Dennard scaling means that improvements in performance and power consumption must now come from additional complexity (e.g., many cores, power geting) rather than relying on technology scaling.[1]

[1]The impact of Dennard scaling was pointed out to me by Prof. Subhasish Mitra of Stanford.

Fact 2: The only technique known to humanity to enable scalable complexity is formalization: This fact is not common knowledge, so it merits some reflection. Consider verification, for instance. For trivially small designs, a few ad-hoc simulations suffices. For a larger design, one would create a proper testbench. When designs became even larger, that led to testbench automation tools, in which testcases become increasingly formally specified. More recently, we have witnessed the rise of assertion-based verification, which is essentially a methodology of adding fully formal specifications to a design, thereby naturally enabling fully formal property checking. Each time there is a need for greater complexity, the only way to manage it is to formalize what had previously been informal.

Note that this phenomenon is not limited to verification. For example, we can see the same pattern in which RTL was introduced first as a semi-formal human notation, in order to better document what were at the time were considered highly complex designs. RTL then became formalized into machine-readable hardware description languages, which enabled greater scalability via automatic (simulation) tool support. To scale further required codifying (i.e., formalizing) a synthesizable RTL, leading to logic synthesis. And so forth.

Indeed, this pattern of scalable complexity enabled by formalization repeats throughout human history. Standardized weights and measures enabled interchangeable parts, which enabled the assembly line and the industrial revolution. Standardized time zones enabled large-scale interconnected rail networks. We can even trace the scalability of markets and the success of capitalism to the formalization of property ownership [7].

The above two facts are sufficient to prove that formalization is a necessary condition for continued progress in SoC verification. It doesn't tell us, however, *when* that will happen, nor is it reassuring to those who dislike formality. Thus, I raise a third fact:

Fact 3: Formalization wins when it is less painful than the alternatives: Formalization requires effort and thought, and most people avoid extra effort and thought as much as possible. Accordingly, although formalization is necessary for scalable complexity, people avoid it until they reach a level of scale that requires it. This is natural and appropriate. Would you use a lawyer to formalize a sales transaction at a garage sale? Probably not. Would you use a lawyer to formalize the sale of a house? Quite possible. Would you use lawyers to formalize the sale of a major corporation? Undoubtedly, and of the highest caliber available.

Similarly, would you use formal verification on a trivially simple design? Probably not (unless it were effortless to use). Will you use it when you have no other way to gain sufficient assurance in the correctness of a complex SoC? Of course. You will have no other choice.

The good news is that while rising complexity is making non-formal alternatives for SoC verification increasingly painful and untenable, research advances and commercial tool improvements are making formal techniques increasingly easy and painless. When formal becomes less painful than the alternatives, formalization wins. For example, consider the rapid shift in the 1990s from RTL-to-gate simulation to formal equivalence checking, as soon as lengthy simulations had become more painful than the increasingly capable formal equivalence checking tools. Or more recently, it was the largest companies with the most complex designs who drove the adoption of formal tools and methodology, but these are becoming widespread as ordinary designs have become similarly complex while the formal tools have become increasingly capable and easy-to-use.

In the future, the same trend will envelope firmware and software development, as well as analog and mixed-signal circuits. Formalization will also enable a rich marketplace for (hardware and software) IP, with buyers having confidence because sellers can formally prove that their products will integrate seamlessly into the buyer's SoCs.

RONNY MORAD, IBM RESEARCH - HAIFA, ISRAEL

Hardware-only system-level verification is definitely a growing challenge. SoCs today have more logic than ever before, encompassing functionality that spans from CPUs, to IO interfaces, message passing bridges, special purpose accelerators, and more.

All this functionality needs to operate together correctly. For example, a CPU must be able to handle the sending of a message while being interrupted because an accelerator completed its work. Therefore, the number of possible interactions between all components in an SoC is huge. What exacerbates this situation is that a bug in a single component in an SoC may require another tape-out, unless it can be easily bypassed by software. This, in turn, creates schedule delays and additional costs.

Due to all of the above, HW-only system-integration has to be verified. Reusing unit-level environments is not sufficient, since each unit is focused on its functionality and is not concerned with the complex interactions at the SoC level. Note, that HW/SW co-simulation does not solve this problem either. There are two reasons for it. The first is that the SW exercises the design in a certain way, and rarely reaches corner cases. The second reason is that when the SW changes, it may interact with the design in a different way, which has not been tested before.

Luckily, the server companies have been dealing with this challenge for quite a while and several tools and methodologies have been developed for the purpose of system-level verification [8], [9]. These tools are characterized by the fact that they are focused on handling the system as a whole. They allow the verification engineers to create scenarios where the stress and the complexity are at the system-level, and not within an individual unit. These scenarios can expose bugs that are impossible or very hard to find at the lower-levels.

I propose to leverage this kind of tools and methodologies for SoC verification. However, the differences between server chips and SoCs present challenges that make it difficult. The first notable difference is that the size of a verification team that works on a typical server chip is much bigger than the size of a corresponding team that works on a typical SoC. That is why it is reasonable to invest the effort required to use the system-level verification methodology mentioned before for a server chip. The second difference is that in a typical SoC

there's much more 3rd party IP, than a typical server chip. So, utilizing this fact may be the key to adapting server verification methodologies for SoC verification.

Another growing challenge in this domain is the fact that the interaction between HW and SW is becoming more and more complex. This is because neither the HW nor the SW alone can satisfy the growing demands for increasing functionality and performance with limited power consumption. The current practice is to document the HW/SW interface using specifications written in natural language (English). However, this creates several problems. Just a few examples include: contradictions within the specification, ambiguity with respect to responsibility between HW and SW (i.e., it's not clear from the text whether HW or SW needs to carry out a certain task), or vague and hard-to-interpret sections.

A possible solution for this problem is to formally specify the HW/SW interface (and interface only). This can prevent some of the problems noted earlier. While there has been some work done on formal specification of HW/SW interfaces [10], [11], it has not yet matured enough to be widely adopted by the industry.

I propose to further pursue this direction as this also enables the automatic generation of FW/OS/device driver code, as well as test case generation for verification purposes.

FRANK SCHIRRMEISTER, CADENCE DESIGN SYSTEMS, SAN JOSE, CA, USA

Multi engine and abstraction verification

SoCs certainly tend to cluster in their architecture topologies around specific application domains automotive, wireless comms, wired comms, graphics, gaming, industrial etc. Within the application domain, starting with a base platform, SoCs more and more often are derivatives of the base platform and can address different performance, power and cost points starting from the same base architecture.

With growing complexity and further increasing challenges in verification which was, is and probably always will be an unbound problem in itself, verification environments need to evolve from the pure unit level to Full SoC and even SoC in System verification that take into account the SoC and system-level aspects including the software executing on processors appropriately. Key aspects to be addressed going forward include

- enabling users to run as many verification cycles as smart as possible using combinations of transaction-level simulation, RTL simulation, RTL emulation and RTL FPGA based prototyping
- efficient hardware/software co-development and co-debug
- verification automation enabling the re-creation of different derivatives of verification environments as efficient as possible
- virtualization of the system environment to efficiently represent the environment in which SoCs reside in
- the emergence of software as an instrument to verify hardware

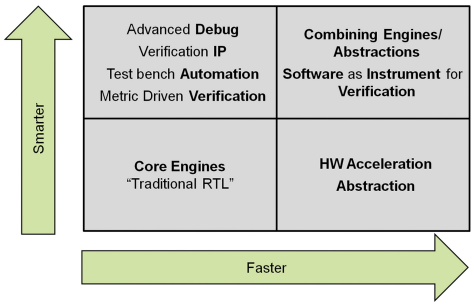

Fig. 1. Faster and Smarter Verification

Only a combination of the various techniques assertions, checkers, coverage analysis and smart connections between different levels of abstraction will allow to keep the gap between verification needs and verification capabilities small enough that users get the confidence level high enough to actually tape out their designs.

Verification Engines

Figure 1 outlines the concepts of faster and smarter verification. Starting from the core engines of RTL simulation, on the horizontal axis methods for faster verification are outlined:

- RTL simulation is the baseline for verification and is characterized through fastest turnaround time once a change in RTL code has been made as well as best in class hardware debug.
- For longer simulation sequences as well as for faster execution that enables software bring-up of operating systems (OSs) like Linux, Android and Windows Mobile, Processor Based Emulation uses hardware acceleration to raise the speed from the Hz or KHz range in simulation to the MHz range. It is characterized through fast compile times on single hosts, allowing the mapping of multiple RTL drops per day, with great hardware software debug.
- To allow speed-levels that are sufficient for software development, FPGA Based Prototyping maps the design into an array of FPGAs and raises the speed level to tens of MHz. In exchange, though, traditional manual bring-up takes months at a time, design capacity is limited and hardware visibility for debug is very limited or intrusive and degrading speed. FPGA Based Emulation positions itself in-between Emulation and Prototyping, with faster bring-up than prototyping, more limited hardware debug and only marginally faster execution speeds.
- In the spirit of not letting bugs slip into the design in the first place, verification can be executed faster at higher levels of abstraction using transaction-level models (TLM [4]). TLM test benches can be re-used for verification of RTL, TLM based virtual prototypes allow earlier hardware verification and serve as verification reference. In conjunction with high-level synthesis, TLM based verification flows are emerging.

Faster and Smarter Verification

To achieve more efficient smarter verification for all engines, Metric Driven Verification, Automation of Test Benches, Verification IP and Advanced Debug features for better root cause analysis can be added. To achieve smarter and faster verification the various core engines can be combined:

- Simulation acceleration is the combination of RTL simulation and hardware acceleration.
- Virtual prototyping and hardware acceleration can be combined to allow accelerated OS bring-up and faster software execution.
- By making software an instrument of verification, test bench development can start on virtual platforms, can be refined on RTL simulation and accelerated hardware execution and even run on the actual silicon as diagnostics, effectively shifting traditional post-silicon verification techniques to the left into pre-silicon hardware/software verification.

REFERENCES

[1] Accellera, "UVM - Universal Verification Methodology," http://uvmworld.org/.

[2] R. Drechsler, *Advanced Formal Verification*. Springer, 2004.

[3] N. Ip and S. Swan, "An introduction to the new SystemC verification standard," in *Proceedings of the 2003 Design, Automation and Test in Europe Conference (DATE)*, March 2003.

[4] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '03. New York, NY, USA: ACM, 2003, pp. 19–24.

[5] "IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tools flows," 2010, IEEE Std 1685-2009.

[6] "MySQL," open source database. http://www.mysql.com.

[7] H. De Soto, *The Mystery Of Capital*. Transworld, 2010.

[8] R. Emek, I. Jaeger, Y. Naveh, G. Bergman, G. Aloni, Y. Katz, M. Farkash, I. Dozoretz, and A. Goldin, "X-Gen: A random test-case generator for systems and SoCs," in *Seventh IEEE International High-Level Design Validation and Test Workshop*, 2002, pp. 145–150.

[9] D. Geist and O. Vaida, "A method for hunting bugs that occur due to system conflicts," *IEEE International High Level Design Validation and Test Workshop Location: Incline Village, NV, USA*, pp. 11–17, 2008.

[10] J. Li, F. Xie, T. Ball, V. Levin, and C. McGarvey, "Formalizing hardware/software interface specifications." in *ASE*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE, 2011, pp. 143–152.

[11] P. Gerin, H. Shen, A. Chureau, A. Bouchhima, and A. A. Jerraya, "Flexible and executable hardware/software interface modeling for multiprocessor soc design using systemc." in *ASP-DAC*. IEEE, 2007, pp. 390–395.