Quarterly newsletter for verification engineers

Issue 3 | 2013

Advanced Verification Bulletin

About This Issue

Welcome to the Advanced Verification Bulletin!

With every leap in design complexity, verification grows in importance. Consequently, the role of the verification engineer becomes more visible and grows more difficult. Greater access to the newest trends and thoughts in advanced verification can play a major part in aiding the verification community

In order, then, to help you grow as a verification professional, we are pleased to present this edition of the Advanced Verification Bulletin (AVB). The goal of the AVB is to provide valuable practice, tool, and trend insights to you, the verification professional, so that you will gain a greater understanding of best practices and upcoming trends in Synopsys Verification.

Inside this issue, you'll find:

- A guest article from Amol Bhinge from Freescale, who shares his thoughts on reducing verification turnaround time for what he calls "monstrous" SoCs...
- An 'Inside Synopsys' view of emerging trends in debug, emulation, functional verification, mixed signal verification and VIP …
- An update on upcoming events of interest to the verification community

We hope that you will find this issue AVB useful and timely!

Regards,

The Advanced Verification Bulletin Team

We welcome your comments, suggestions, and topic ideas, so feel free to send us your feedback at **avb@synopsys.com**.

Reducing Verification Turnaround Time for "Monstrous" SoCs

Freescale's Amol Bhinge discusses how strategic verification planning is essential to meeting the exploding SoC verification challenge.

Using the word 'complex' to describe SoC's is far too mild. Industry-wide, our chips are becoming monstrous.

For example, I manage the group doing high-end SoC verification for Freescale's Digital Networking Group, and we recently taped out the T4240 processor, which had more than two billion transistors. Following Moore's Law, our next-generation chip may have more than four billion transistors.

In addition to sheer chip size, the variety of functionality on SoCs is also growing. Expanding functionality on each SoC is causing verification complexity to not just double, but grow exponentially. To continue my example, the T4240 had 24 virtual cores, 12 dual-threaded physical cores, more than 50 functional blocks, and over 100,000 lines of testbench code. These 100,000 lines of testbench code translates to 14,000 individual testcases, with over one million lines of testcase code.

This is in line with what our colleagues in the communications industry are also taping out. And we can all say that it's truly monstrous.

In This Issue

But despite gatecount doubling to follow Moore's Law, verification teams don't get double the resources, nor do they get double the time. If anything, it's the opposite. Consumer demand drives our schedules and cost structure lower and lower. It's the electronics ecosystem that drives us all to more complexity, more cost controls, and tighter schedules.

In this environment, it's not physically possible for verification teams to check absolutely every combination of signals that could go through an SoC. Yet, there is no way to compromise on quality. Unlike the software business, where a company can ship a patch if they find a post-production bug, in silicon, the product has to be right the first time.

SoC verification becomes a complex equation that depends on the four complex variables: schedule, resources, complexity and quality. Balancing this equation involves some science, but it is also something of an art, because there are no commercial tools to help you decide what not to check. It sometimes requires buy-in from the entire business unit to agree on leaving something off the verification table. And, with this complicated equation, the SoC verification team ends up providing the last line of silicon defense.

One way to understand this last line of silicon defense is to think of the verification team's job as being comparable to airplane security. In the US, there might only be one security check before getting on an airplane. But in much of the rest of the world, there are two or three checkpoints. It's the final checkpoint that has to do the most thorough check, because if there is something in a handbag or a briefcase there's no one else to stop something bad from happening on the plane. It's the final checkpoint at the end who has to make sure that everyone else has done the checks correctly. But he can't take so long that the plane doesn't leave on time. He has to be strategic in who and what he checks, while being highly thorough.

With chips today, it is the SoC verification team that does that final strategic checking. If there's any bug in the design, the SoC verification engineers are the ones who have to find it, even if the IP verification team already checked that block.

But if SoC verification teams are going to have the visibility needed to be effective as that last line of silicon defense, minimizing verification turnaround times is critical. It allows us to virtually expand the 'resources' part of our verification equation without having to add engineering time. The faster a team can turn around effective verification, the more detailed the team can get with its verification. But minimizing turnaround times not only requires high performance simulation, it also requires proactive strategy and detailed planning.

For example, on our "T4240" project, we really needed to reduce our simulation compile times. On the previous generation chip, our incremental compile times took approximately an hour. With double the complexity, we were worried that it would take us more than two hours to do each incremental compile. If it did, and we had to recompile our environment every time a testcase was changed, my team would only be able to do a couple of iterations per day. That wasn't going in the right direction.

We proactively worked with Synopsys to adopt the VCS Partition Compile methodology. Using Partition Compile, a verification team can partition a design and its testbenches into discrete sections. Then, VCS will only do an incremental compile on any section that has a change. By planning out the design, and creating these partitions, our verification team shrunk what was an hour-long process with our last generation chip down to a 28 minute process on a chip that was twice as complex. And, at the gate level, it reduced simulation compile times from days to hours. It was a great first step to managing our SoC complexity.

Similarly, it takes a strategic approach to reduce simulation runtime. Simulation times for complex SoCs can run for days, and verification teams run thousands of simulations when checking an SoC. Shortening simulation runtimes can have a huge impact on the overall verification schedule.

Typically, whenever anyone talks about slow simulation runtimes, users want to blame the simulator. But with SoC complexity, it's not enough to improve the speed of simulation to get the reduction we need to meet our complexity demands. Users have to be more strategic in how they actually use verification products.

For example, instead of running designs with real cores, users may use virtual cores to start checking interfaces earlier in the process, and do more IP verification at the subsystem level; users can review legacy testbenches to try to eliminate duplication of testcases and reduce simulation runtime without sacrificing quality; or they may enable only the VIP components that are needed for any given simulation. Any of these approaches can reduce simulation runtime without sacrificing verification quality—but only through strategic, detailed, proactive planning so that the verification team is working with the right criteria.

It is also important to get very good, detailed data on where a team is spending its verification time, and use the versions of the simulation solution that will give that data.

Again, using the T4240 as an example, the verification team was able to use the new, advanced Profiler in VCS to learn exactly how VCS was using its time. The Profiler found that VCS was spending 52% of its

time on constraints, when there weren't a lot of constraints at the SoC level. Our team used a miraculous VCS option (switch) to improve constraint solver performance. Using that option brought a 12 hour simulation runtime down to 2 hours.

Across the industry, there is often inertia in adopting the latest and greatest versions of our EDA software. Verification teams, in particular, are scared of breaking testcases or ruining the verification environment. And, this is an understandable reluctance, since proven testbenches are the bread and butter of verification teams. But without leveraging the thousands of hours that our EDA partners put into updating the latest versions of verification products—with their improved functionality, commands, VIPS, and utilities—verification teams aren't going to get the 2 to 6X improvements in verification turnaround that are needed to keep up with complexity.

When I look to our next generations of chips, I still see lots of room for innovations that will allow us to improve turnaround times and continue to balance out or complex verification equation. Improving debug strategy, making it easier to manage memory footprints, improving gate-level verification, and building the ability to reuse testcases throughout the design flow are a few examples.

In addition, there's a lot of room to improve state space management. Today, when we look at our state space, an SoC verification engineer has millions of different possible combinations to look at. It sometimes takes the entire company to figure out where to start looking and what to exclude. Today, the industry is still putting together a solution to effectively help us look through that process, and that is an area where our verification vendors can help us in the future.

Ten years from now, following Moore's Law, our chips are going to be more than 30X more complex. If we are going to be around as an industry, we need to solve these scalability problems proactively, rather than reactively. We're going to need best-case simulation and verification environments. I anticipate that we will be doing more and more work through third party IPs, using standards wherever possible, and letting our companies focus their differentiation on SoC design and verification.

And, by doing these things, verification teams will continue to ensure that our chips, like our planes, will be safe as they leave our gates and go out into the world, for we will be able to continue to tame and control the monsters that will be our SoCs.

About the Author

Amol Bhinge is a senior SoC verification manager at Freescale, responsible for high-end SoC verification for the Freescale Digital Networking business group. He has been actively involved in industry standards efforts, was a member of the Accellera UCIS-TSC that rolled out UCIS 1.0, and is a regular contributor to the Synopsys Users Group Conferences (SNUG). He holds an MS degree in Electrical Engineering from the University of Minnesota and has 7 US patents related to verification tools and methodologies.

Amol Bhinge
Senior SoC Verification Manager,
Freescale Semiconductor, Austin, Texas, USA

Functional Verification

With each generation of SoC, gate counts are scaling by a factor of two, but the number of possible interactions inside and between functional blocks goes up exponentially, meaning verification complexity is rises exponentially. Constrained random verification leverages randomness to enumerate all possible stimulus scenarios, while constraints strategically focus that stimulus onto the most critical verification requirements. This combination leads to far better coverage in less time than traditional directed verification and is absolutely required to address the larger and more complex systems that are now being produced, which is why it is being widely adopted by the industry.

Yet, the question then arises, with 50–100 thousand lines of constraint code, coming from a combination of legacy IP, third party IP and newly created IP, how do you ensure the quality of constraints, and minimize the possibility of conflicts?

Rebecca Lipon
Product Marketing Manager for Functional Verification

Analyzing Constraints to Improve Stimulus Quality with VCS

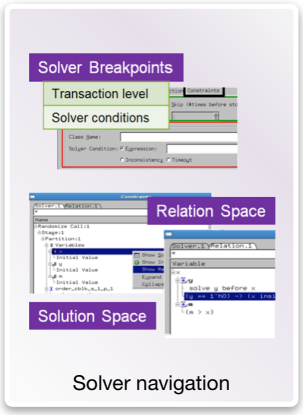
Constrained-random verification is a fact of life when verifying today's complex chips. SoC verification environments now contain 50–100K lines of constraint code, coming from a variety of IP sources with different coding styles. As a result, there can be conflicts or challenges within the constraint hierarchy that can lead to a simulation not performing as expected.

Traditionally, analyzing constraints has used a time-intensive text-based batch-mode process. Although it is highly automated in VCS, the batch mode process may involve one or more simulation runs. Lengthy solver traces can be difficult to process and the iterative debug process requires additional compile and run time.

With SoC verification complexity, this process needed to be sped up and made easier for the user. The next-generation constraint analysis tools in VCS 2013.06 address this with a GUI-based, interactive, one-pass debug approach consisting of constraint navigation, interactive debug with on-the-fly re-randomization, distribution analysis, and interactive testcase extraction.

Constraint Navigation

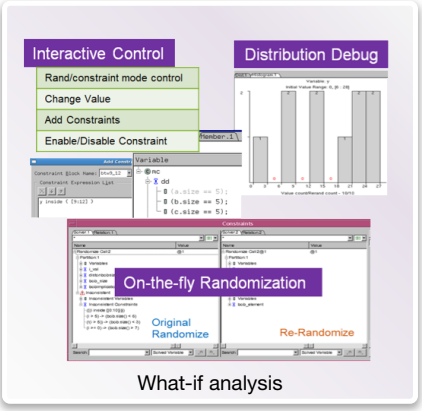
Constraint navigation enables a user to look at the interactions of different constraints throughout a simulation. The location of the randomize call may not be immediately obvious either because it is embedded in an unfamiliar IP, or even within a methodology base class such as UVM. In batch mode, the origin of a constraint result would be difficult to analyze or understand, but with the flexible, GUI-based constraint navigator, users can search views of solver traces, identify variable relations, and cross probe to different views of the debugger (class browser, local pane, and source) to root cause any constraint problem.



Solver navigation

Interactive Debug

This GUI-based approach also enables interactive debug without needing to re-compile. A user can interactively edit constraints, re-randomize based on the changes, and see the outcomes without any additional simulation compile or run time. The user can also look at the solution distribution of the given constraint problem to preemptively analyze the quality of the stimulus that the constraints intend to describe—whether it's in the configuration of constraints or transactions generated by the stimulus. This analysis is enabled through features like interactive constraint editing, on-the-fly randomization that doesn't require recompilation, and distribution debug.



What-if analysis

Figure 1: GUI-Based Interactive Constraint Debug and Analysis

Debugging an inconsistent constraint offers an example of how the new interactive debugging capability might work. If a randomization call has no solution, then somewhere in the sea of constraints there are contradictions. To solve the problem, VCS automatically narrows in on the minimal set of constraints that are causing conflict. Then, the user can open DVE and use its new, interactive debug capabilities to quickly pinpoint the problem.

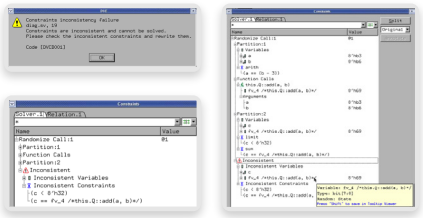


Figure 2: Debugging Constraint Inconsistency

After some analysis, the user may identify that the inconsistency can be resolved by adding some constraints. Without leaving that debug session, the user can interactively add new constraints to the randomization.

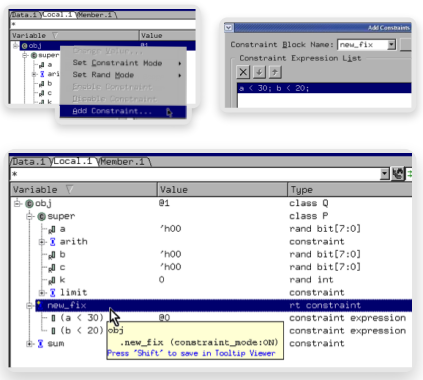


Figure 3: Interactive Constraint Editing without Additional Compile and Run Time

Now the user can immediately see the resulting solution space from the modified constraints by executing a re-randomization on-the-fly—without recompiling the design.

This GUI-based debug approach also simplifies the resolution of unexpected solutions (which often represent nasty bugs hidden deep inside a testbench) and soft constraints.

Distribution Analysis

These interactive debug features can also be used to do distribution analysis to verify the quality of the stimulus before coverage is even written. Within the GUI, a user can perform multiple re-randomizations with different random seeds to analyze the solution distribution of the randomization. If there is a less-than-ideal solution distribution, the user can interactively edit the constraint model, adding, editing, or deleting constraints to analyze their impact as many times as required until a desired distribution is achieved. All of this is done without re-compiling or rerunning the simulation, translating into significant time savings.

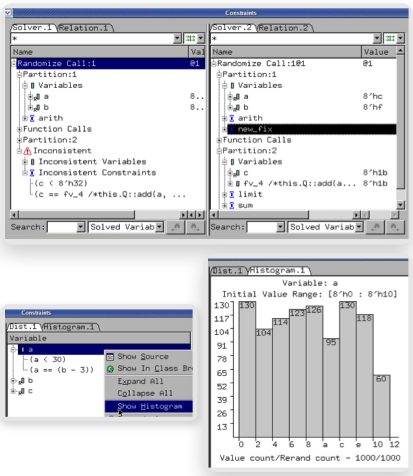


Figure 4: Interactive Solution and Distribution Analysis for Randomizations

Testcase Extraction

For more complex constraint problems involving multiple random variables and constraints, it is often useful to have VCS write out a standalone testcase for further analysis. In the batch mode debug solution supported in previous versions of VCS, we could do testcase extraction, but we would need to rerun the simv again with an extract option. This extraction process could take a long time if the randomization being investigated required significant simulation time. With the GUI-based constraint analysis tools, this re-run of the simv with an additional option is unnecessary. At any moment, the user can extract a testcase on demand within the GUI on the specific randomize call currently being debugged, saving multiple simulation iterations and cutting out hours of runtime.

Improving Stimulus with VCS Constraint Analysis

The interactive constraint editing and on-the-fly re-randomization capabilities in VCS can significantly increase user productivity when debugging constraint related issues because they require no recompilation of designs and restarting the simulations. Combined with distribution analysis, these features enable pre-emptive analysis during constraint development which will ensure that the desired solution distribution and coverage from constrained random verification can be achieved sooner.

There are many more constraint analysis capabilities in VCS 2013.06. For more detailed information on new constraint analysis solutions in VCS, please see Jason Chen's webinar at http://tinyurl.com/lmsv2n5

For a large design, capturing emulation waveforms can be a long and difficult process. Potentially millions of cycles of full visibility debug data must be transferred from emulation trace memory to a PC, where it is then processed to generate waveforms. On many emulation platforms, the engineer is forced to wait an hour or more for this processing to complete before any waveforms can be viewed. And, that's with the help of a farm of workstations.

To improve this time-to-waveform challenge, ZeBu employs a new, interactive, Combinatorial Signal Access (iCSA) technology. With iCSA, a user can select which signals they want to see first, and can usually start debugging within a few minutes. Best of all, iCSA is integrated with the industry's leading Verdi[3] debug environment (see Figure 3), which means that most verification engineers are already familiar with this environment (and only one PC is required). The best part: while the engineer is debugging the first set of signals, iCSA and Verdi silently generate the remaining signals, meaning the next set of signals the engineer selects may take only a few seconds to view. Not only does this technology make debug across billions of cycles practical, it does it while providing 100% full-node visibility.

Lawrence Vivolo
Product Marketing Director for Emulation

Using a Simulation-Like Debug Methodology Inside an Emulation Environment

In today's SoC design environment, many design teams are turning to emulation systems for system-level hardware/software (HW/SW) co-verification. Increasingly, however, design teams are finding the debug limitations of most emulators are frustrating their verification efforts. Large SoC verification now requires tests spanning billions of cycles (see Figure 1). Traditional emulation systems with only logic-analyzer style debug tools have limited scalability and are no longer sufficient.

Now, Synopsys ZeBu® goes beyond providing traditional logic analyzer debug—combining the best of traditional emulation debug with the ease of use and determinism of RTL simulation—to provide a 100% visibility debug environment that spans billions of cycles. Focused on debug throughput, ZeBu delivers maximum performance, both runtime and debug.

Traditional Emulation Debug vs. Simulation

Today, over 80% of all bugs are found with traditional simulation, and virtually all verification teams continue to rely heavily on simulation. With simulation, the typical use model is to simulate test cases logging any failures with assertions and monitors. The verification team then reruns any failing tests to debug the failures using a standard debug environment such as Verdi[3]. Ideally, such a use model includes assertions, monitors and protocol VIP that extends directly to the emulation environment. This increases ease-of-use and avoids the necessity of introducing a different debug methodology.

Unfortunately, this type of environment is not readily available in emulation environments, especially those using in-circuit emulation for verification where the design and its verification environment are principally implemented in physical hardware. Unlike simulation, emulation environments are rarely deterministic; requiring all debug data to be captured on-the-fly. So users are typically forced to reply upon an emulation system's integrated logic analyzer, where they set up watch points and trigger conditions, and hope to capture all the information required to debug the failure in trace memory. While some emulation platforms do offer full

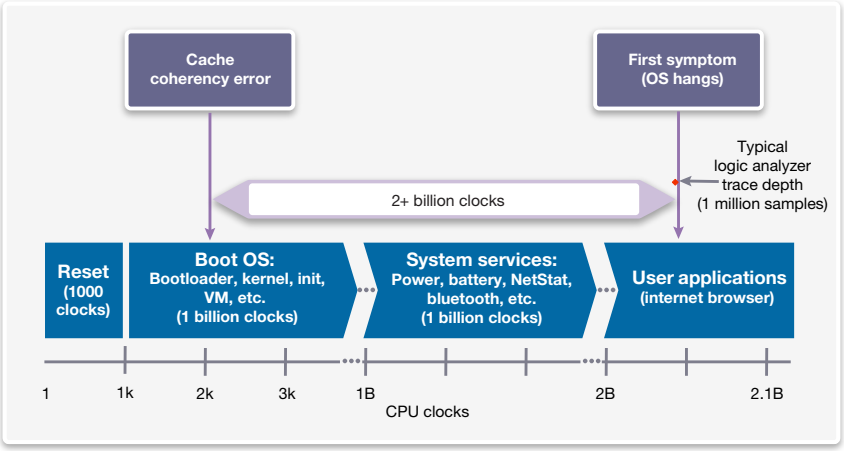


Figure 1: Cache coherency failure spanning billions of cycles

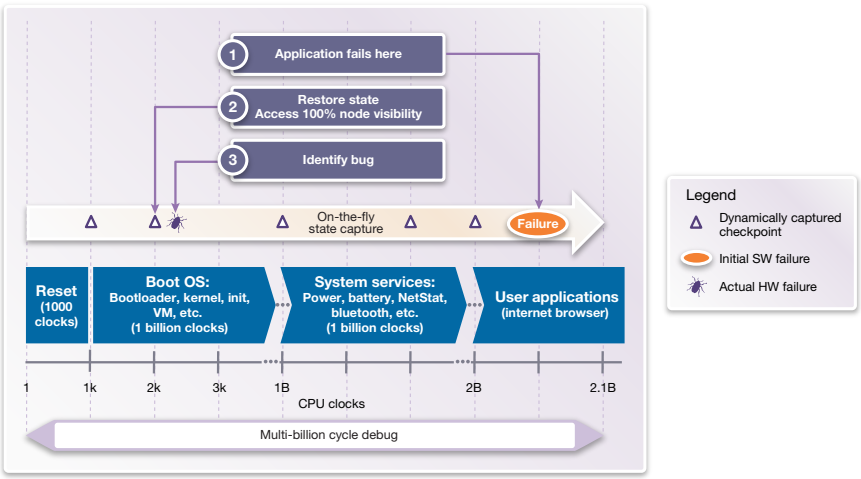


Figure 2: ZeBu post-run debug flow

visibility with their logic analyzers, this visibility is limited to the depth of the logic analyzer's trace window. And, here is where the challenges arise.

First, most logic analyzers used in emulation today (integrated or not) are limited to a trace depth of a million cycles, or so. Even the best available logic analyzers cannot span billions of cycles with 100% full-node visibility into the design. So, while these logic analyzers are great for capturing early RTL bugs, such as reset errors that occur hundreds of cycles into a given test, they are frustratingly limited when tracing system-level HW/SW integration bugs that manifest themselves billions of cycles after the actual problem occurred. Because of the trace depth limitation, users find themselves constantly editing the trigger conditions, and rerunning tests in an effort to try to capture events earlier and earlier. This brings us to another fundamental challenge: lack of determinism.

With traditional emulation systems, especially while running in in-circuit mode, there is virtually no determinism available. Since events are somewhat random, failures can vary from run to run, thwarting engineering efforts to trace a specific problem. As a simple example, imagine booting a tablet computer using in-circuit emulation where there's a flaw in the CPU's cache coherency logic. In one case, the bug corrupts the executable for the internet browser early during the boot cycle. Several billion cycles later, the browser is launched and the program crashes. Since the trace buffer was only a few million cycles deep, the engineer cannot see the original cause of the browser crash and is forced to change the trigger conditions, rerun the test, and try to capture the bug earlier in the test. Unfortunately, because of the randomness of events, something else in the cache is corrupted on the second run. As a result, a different program crashes, and the trigger conditions are no longer valid.

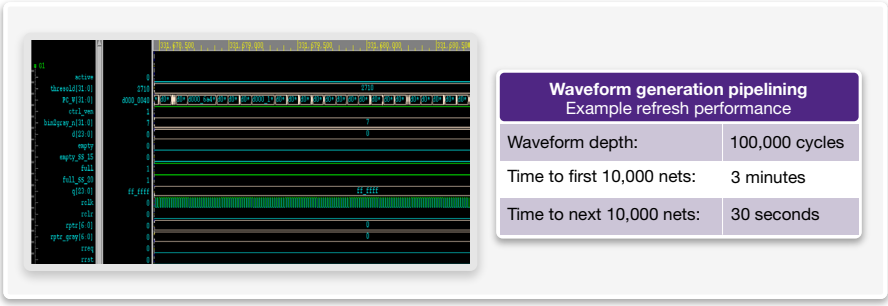
While the ZeBu emulation system supports a logic analyzer-style methodology, most ZeBu users quickly recognize the additional power that its simulation-like debug methodology offers.

Using a Simulation Debug Methodology Inside Emulation

ZeBu has the capacity to support the same methodology that every engineer has used in simulation.

First, verification engineers transfer their existing simulation environment, complete with assertions, monitors and VIP protocols, directly into ZeBu—effectively making ZeBu an extension of VCS or another leading simulation tool.

Then, the verification engineers can kick off a testcase, let it run for hours or even days until it fails. Once a failure occurs, the user need only review the log files, just as is done in simulation, to decide where to debug. ZeBu has already captured all the data required to generate 100% full visibility data, no matter how many billions of cycles have transpired. And since ZeBu is fully deterministic, engineers can replay any test knowing that they will always see the exact same results, eliminating seemingly random error chasing (see Figure 2). This provides ZeBu the deterministic verification environment of simulation, the 100% visibility debug of a traditional logic analyzer, and the billion-cycle capacity of an emulation system—all of which are needed together for system-level SoC debug.



Waveform generation pipelining Example refresh performance	
Waveform depth:	100,000 cycles
Time to first 10,000 nets:	3 minutes
Time to next 10,000 nets:	30 seconds

Figure 3: Interactive waveform generation with Verdi[3]

Using HAPS to Streamline IP to SoC Integration

Today's ASIC IP and SoC design teams face the dual challenge of very short delivery schedules and high risk of their product being rejected by the market if chip designs ship with defects. Synopsys' FPGA-based prototyping solution enables a more parallel hardware/software development strategy where software developers, validation engineers, and system integration experts have access to prototypes running at near real time speed months before tape-out of new ASIC silicon. FPGA-based prototypes are ideal for pre-silicon software development, system validation, and hardware/software integration of ASIC IP and SoC designs. Prototyping teams benefit from an easy-to-use flow from ASIC RTL to the FPGA-based prototype, high debug visibility, and a scalable hardware architecture to support and streamline IP and SoC system level validation.

Neil Songcuan
Senior Product Marketing Manager
FPGA-Based Prototyping

Today, approximately 70% of ASICs are using FPGA prototyping for system validation. The increased popularity of FPGA prototyping has two significant drivers: the increase in software elements such as applications, drivers and firmware that need to be developed and validated in SoCs, and the hardware complexity of combining IP from a variety of sources, multiple processors and high-speed external interfaces, such as USB, HDMI, and PCIe.

To maximize the time available for software development, while still providing the necessary time and bandwidth to validate silicon components, Synopsys has architected the HAPS-70 series FPGA-based prototyping platform to support early IP validation through IP to SoC integration.

Traditional FPGA-Based Prototyping Board Product Architecture

Traditional hardware prototype boards rely on a structured architecture with predefined interconnects that limits a user's ability to do early IP validation, and, then port that IP to a full prototyping system.

To use an analogy of how this architecture works, imagine that the FPGA is set up like a large city. Each logic, functional block is a complete, self-defined neighborhood connected to other neighborhoods, but only by freeways that were constructed before the neighborhoods were built, or even fully planned. Each freeway might have space to carry hundreds of signals between each neighborhood, but they aren't flexible, moveable, or expandable. This might work as the neighborhoods were first built, but if everyone moves to one neighborhood and works in another, suddenly that arterial freeway is going to be jammed with traffic—while others are completely empty—and there's no easy way to divert the traffic, or expand the freeway.

Similarly, in traditional prototyping systems, there are prebuilt interconnects between blocks. So, for example, there might be 200 interconnects between block A and block B, and 200 interconnects between block B and block C. The user defines the logic in these blocks, but not the interconnects. So, when the next generation chip needs 400 signals between block A and block B, there's no way to shift the interconnects between block B and block C, and no easy way to integrate an IP prototype if the pins don't align.

HAPS-70 System Architecture

With HAPS-70, Synopsys provides a dynamic, configurable, symmetrical architecture that offers support from the IP to SoC level. Instead of prebuilt interconnects, the HAPS-70 architecture has flexible user-defined I/O ports that are suited to address changes from design-to-design. This allows SoC validation teams to re-use the HAPS-70 system for varying design styles, and from project-to- project.

The HAPS-70 architecture offers a completely, symmetrical architecture between its smaller S12 system, which supports up to 12 million ASIC gates, up to the larger S144 product, which supports up to 144 million ASIC gates. HAPS-70 products have identical pinouts, so the S12 can effectively be stitched into the larger system with complete reuse. With over 1,000 I/O ports on each S12, users have near infinite flexibility in configurations.

In this case, instead of designing a city where neighborhoods have to be built onto predefined freeway spaces, you have a completely open system with fully, configurable, interconnects.

This means IP developers can create smaller FPGA prototypes using the S12 system and validate any software drivers, applications and firmware specific to

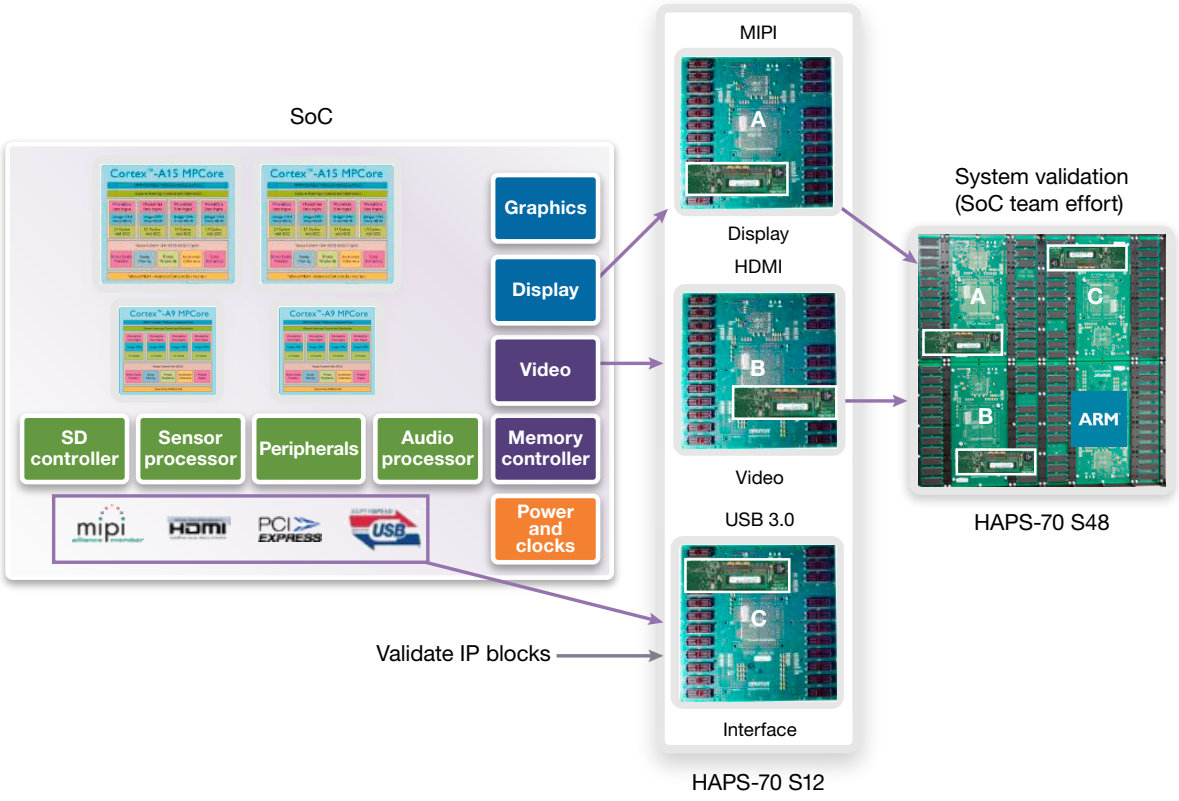


Figure 1: HAPS Architecture Streamlines IP to SoC Integration

that IP. For example, with the design in figure 1, which contains multiple IP blocks from third party vendors or legacy designs that are already fixed, high-performance, external interfaces based on standard protocols, and multiple ARM processor cores, users would do early IP validation to check MIPI, HDMI, and USB 3.0 in parallel.

When the individual IP validation is complete, the implementation and scripts used to optimize and validate the IP can be reused into a larger HAPS-70 system for the full SoC level FPGA prototype. The SoC validation team can then focus on interconnects between various blocks at the SoC level, as the individual blocks will have already been validated—dramatically lowering bring-up time.

With the capability to support IP- and SoC-level designs for system validation, customers will have a common hardware platform available instead of a mix of hardware prototypes that may only be used for specific projects.

The HAPS-70 system, today, offers large capacity FPGAs that are suitable for large blocks of IP or processor cores. It offers configurations that will support up to three S48 systems (supporting 48 million ASIC gates apiece) or one S144 system.

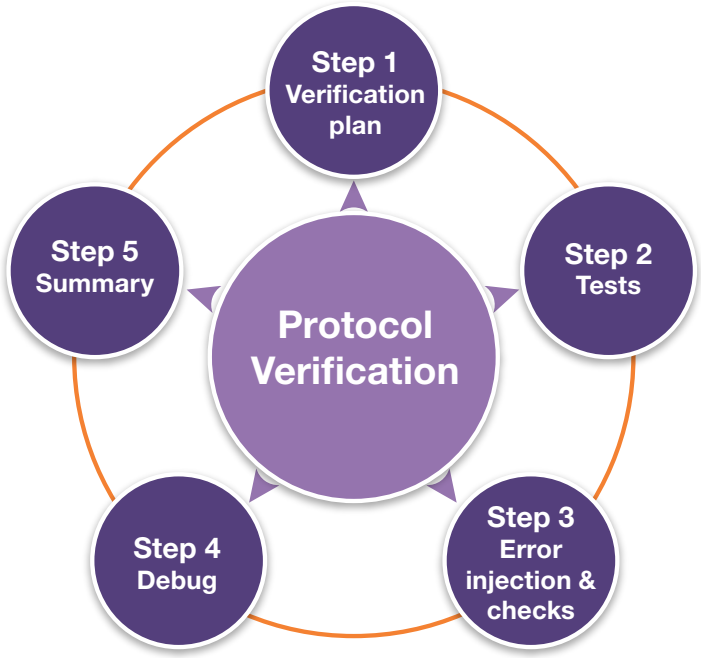
However, many design teams could have multiple, smaller, IP blocks that don't need all that power. The system also has the capacity to link to smaller, FPGA prototypes for smaller, IP blocks, if they are designed to fit into the configurable architecture.

As a result, hardware/software design teams looking to maximize software development time without sacrificing silicon or system quality should look to the flexible architecture of the HAPS-70 system.

One of the main testbench building blocks for building a protocol test environment is high-quality, high performance Verification IP for each of the protocols in the design (as described in the "10 things you need to know about VIP" whitepaper). Verification IPs of the past were simple bus functional models (BFM's): a protocol engine that replicated the behavior of the protocol and generated errors if something at the interface was outside its definition of the protocol. Today's protocol Verification IP is generally much more sophisticated and includes the features needed to fully accomplish and accelerate the five key steps of verification. After-all the engineers that develop VIP are experts in the protocols they develop, so they should embed the benefits of their knowledge into the VIP.

Neill Mullinger
Product Marketing Manager for Verification IP

Five Key Steps of Protocol Verification



A new project is ramping up that includes several protocols and a key task is to ensure that the IPs for the external interfaces are working as advertised, are correctly integrated into the design and support the protocol specifications as required by the product.

There are five critical steps to accomplish this task—build a test plan, develop and run tests to achieve coverage goals, check for and highlight errors, debug unexpected behavior, and extract coverage to compare against the plan. This whitepaper presents key contents of a modern-day VIP that help accomplish the five steps of protocol test.

Step 1: Verification Plan

The five critical steps are iterative until achieving coverage goals but the verification plan is the place to start since it defines the verification goals. The verification plan should fully describe the range of tests needed to verify the functional behavior and verify that a protocol is working as per the specification. It essentially describes the coverage goals and is the metric against which you can measure progress. Today's VIP should include a coverage plan that includes each cover-point for the protocol and includes the cross references to the relevant pages of the specification. Of course it must be user-modifiable to exclude any parts of the protocol not utilized in the design or to add any customizations.

Step 2: Tests

The VIP should include a comprehensive library of sequences from which a user can rapidly assemble tests to model all aspects of protocol behavior. The sequences should at least cover the fundamental protocol behavior needed to verify the interfaces are functioning correctly and comply with the protocol. They should rapidly drive activity that hits the coverage points in the design. For example, a USB port might require sequences that model suspend resume, suspend remote, or suspend connect/disconnect behavior under different conditions. Another general example would be read modify write. Where compliance checking is desired, perhaps for an internally developed IP block, test suites can provide directed testing complementary to the sequence library.

Step 3: Error Injection and Checks

VIPs have become more sophisticated with the inclusion of error injection and expanded error checking. Error checking throws protocol errors at the DUT to make sure it responds appropriately. Does the system respond and recover as per the specification? Checking capabilities go far beyond unexpected behavior that might cause a BFM to error-out to fully cover the validity of the protocol. For example, a sophisticated ARM AMBA ACE VIP uses internal models of the cache to check for data integrity and cache coherence across an entire interconnect. It looks to see if performance metrics defined by the system architects are being achieved. It will ensure that the rules defined in the spec are being followed. It will include all of the checks defined in the ARM ACE specification.

Step 4: Debug

Engineers frequently need a deep knowledge of the protocol to figure out the root cause of errors. As protocols have become more hierarchical, interleaved and complex, understanding the relationships between the layers of the protocols can be extremely challenging. Most VIPs have sufficient error messaging to help identify the symptom of a problem but mapping that back to the log files and waveforms generated in simulation to identify the root-cause can be a time-consuming challenge requiring deep protocol expertise. Synopsys has developed a protocol-aware debug environment that solves this by showing a high-level view of the protocol behavior that highlights parent-child-sibling relationships. It brings together in one place the log-files, the transactions, the object field information, the waveforms and the documentation so that engineers can navigate quickly from error messages to traffic to root-cause. It is simulator independent and can be used to easily share information between verification team members. It is integrated with Synopsys Verification IP and can be easily extended to work with a customer's internal VIPs.

Step 5: Coverage

Coverage combined with the test plan ultimately tells when verification is complete. Prior to that its role if to identify what else needs to be tested, after all there is little point in adopting constrained random verification unless progress can be measured; if it can't be measured, and documented, it didn't really happen, right? Without coverage, it is virtually impossible to find and focus on those tricky corner cases that help achieve code coverage, signal coverage and transaction coverage.

Summary

Your choice of Verification IP goes far beyond how well a BFM models the protocol. A Verification IP should firstly align with the testbench methodology but equally as important offer the tools to rapidly achieve the five critical steps of protocol validation.

Achieving a Synchronized Hardware/ Software Debug Methodology

In today's software-heavy SoC's, it has become virtually impossible to separate software verification from hardware design and verification. Software has proliferated throughout the system with verification-enabling software that has been compiled and loaded onto an SoC's memory model.

With the average SoC today containing multiple processor cores, multiple memory modules, and multiple logic blocks; this is proving to be a debug nightmare. Often, there are multiple processor cores operating in a modern SoC; without the visibility into the software behavior and its related hardware interactions, it is becoming extremely difficult to debug SoC designs. Therefore, a new solution that offers simultaneous hardware and software views will be necessary to enable full SoC debug.

Thomas Li
Product Marketing Director for Debug

It's become virtually impossible to debug the current generation of SoCs without connecting the hardware and software verification processes. (See Sidebar).

However, identifying the exact status of software executing on an embedded CPU has remained a blind spot for achieving full SoC verification and debug. Without a clear path to connect a simulation waveform to a software executable, identifying the best debug solution between hardware and software has proved elusive.

Synopsys has addressed this challenge with Verdi³ HW SW Debug: an instruction-accurate embedded processor debug solution that offers fully synchronized views between hardware, as RTL or gate-level design models, and software, as C or assembly code—enabling co-debug between and SoC's RTL and related software.

Fully Synchronized Views Support Multi-Core Debug

Verdi³ HW SW Debug offers fully synchronized hardware and software debug by connecting an Eclipse-based software view (or Programmers View) with the traditional Verdi hardware view: thereby linking software commands to specific simulation waveforms in real time.

The Programmer's View includes the full spectrum of views typically used by software engineers in sequential instruction flow. This enables software engineers to perform standard debug actions such as: set break points, stop at break point, and stepping through the source code.

The Programmer's View is fully linked with the Verdi³hardware debug views, so that the two windows synchronize in real time. When a user points to any specific hardware behavior, through the simulation time-based synchronization, the specific software statements are highlighted

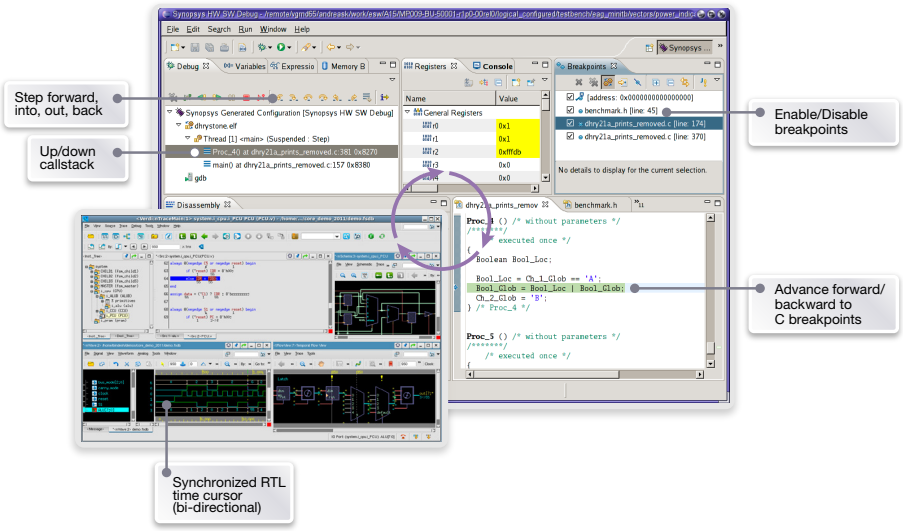


Figure1: Fully Synchronized Hardware and Software Debug Views Support Key Functionality for Software Debug

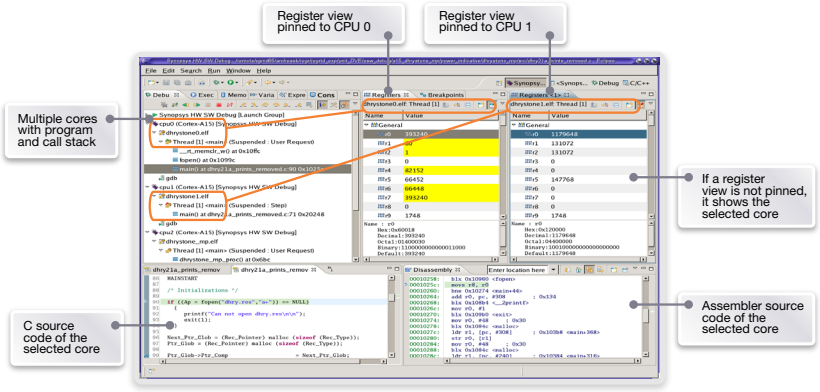


Figure 2: Multi-Core Debug Support

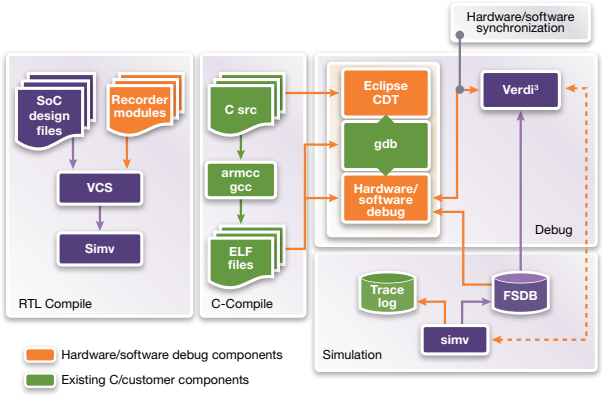


Figure 3: Verdi³ HW/SW Debug use model

inside the Programmer's View. Similarly, tracking any specific software behavior can link back to specific hardware areas.

Modern SoC's contain multiple CPU cores that are often used simultaneously for multi-threaded applications. This means that they must also be verified and debugged simultaneously with sequential stream software commands. However, it has been almost impossible for SoC designers to correlate the simulation waveform and exact software instructions running in the core.

Verdi³ HW SW Debug, on the other hand, automatically establishes the correlation between the waveform and executed software instructions—offering visibility into each core in a multi-threaded path. It attaches all debug views, including the register view and source code views to each specified core. (See Figure 2)

Using Verdi

Setting up and running the Verdi Hardware Software debug software in a simulation environment is a fairly straightforward process.

During the simulation compile phase, the user adds the processor execution trace recorder modules, provided by Synopsys, into the design database. These recorder modules automatically extract the required debug information and save it into FSDB during the latter simulation phase. These recorder modules are compiled along with their rest of SoC design and testbench into the simulation database.

Next, the c-compiler for the targeted processor compiles the embedded program that will be executed for the SoC verification and outputs an ELF file.

Third, simulation runs with the embedded software image loaded into the memory model of the SoC. The resulting waveform from the RTL design and the trace log for instruction execution history will be saved into the fast signal database (FSDB).

Finally, after the simulation is finished, Verdi³ uses the FSDB and the ELF file from the C-compile phase for debugging. While the user is working on the programmer's view, Verdi³ HW SW Debug module will fetch the necessary core information from FSDB and feed them to gdb to replay the simulation behavior.

This flow works with any simulation environment that supports FSDB dumping, and any processor supported by Synopsys.

Verdi³ HW SW Debug is modularized to support an open debug platform. Its data storage is based on the industry de facto standard FSDB. FSDB is supported by multiple simulation environments, emulation solutions and other verification software, enabling Verdi³ HW SW debug to be simulation or verification engine neutral. It also enables Verdi³ HW SW Debug to fit into a wide variety of existing verification flows—from simulation to emulation, and beyond.

For example, users leveraging the emulation speed of the Synopsys Zebu emulation system can dump necessary information like PC and registers into the FSDB. Verdi³ HW/SW debug can then hook into the emulator flow using these references, so that emulation users can do post-process debug based on recorded FSDB.

Conclusion

With the number of processor cores and increase in software-driven applications on complex SoCs, the importance of synchronized hardware and software verification strategies is only going to increase with future chip generations. With the new Verdi³ HW SW Debug capability that integrates into standard verification flows, supports synchronized C, Assembly Language, and RTL views, and enables multi-core debug, users can have confidence that they will be able to debug complex SoCs.

Analog Mixed-Signal

With the proliferations of touch-screens, motion sensors, and real-time control systems, electronic systems must now incorporate an increasing amount of real-world analog phenomena, such as light, sound, temperature, motion or pressure. Designing a system that can read and process this information requires real-time sensor relays such as data converters, amplifiers, RF technologies, or micro—electromechanical systems (MEMS). Each time a user activates the touch screen on a smart phone, looks for accurate readings on a medical device, tracks planes on radar, or reads the average miles per gallon on a car—some kind of analog input must be translated into a digital signal or data for processing in the electronic system or accurate measurement.

At the same time, these systems are often facing stringent battery-life and energy efficiency requirements, calling for advanced low-power circuitry that also presents analog design challenges.

In the past, analog system elements or power supplies may have been contained on standalone chips and verified separately from digital components. But, as leading edge process technology has enabled entire electronic systems to be implemented in SoCs, these components now reside on the same chip and must be coverified. As a result, the once application-specific process of analog-mixed signal (AMS) cosimulation is a necessary part of every SoC design process.

Rebecca Lipon
Product Marketing Manager for Functional Verification

Digital Supplies are Analog

Cosimulation between analog and digital circuitry has always been complex, but smaller process technologies and advanced power requirements have exploded this challenge for every design team.

Virtually every SoC designed today contains analog components in order to incorporate real world information, requiring an expansion in the amount of analog/mixed signal verification (AMS). (See sidebar.) But these SoC's also must contain advanced power circuitry in order to meet stringent battery life and energy-efficiency requirements. This combination of effects requires a new power-aware, analog-to-digital or digital-to-analog verification methodology.

Traditionally, digital HDL's have had no way to model the complexity of low-power environments, as VHDL and Verilog both assume that power is always on. This is why industry experts formed an Accellera committee to create the Unified Power Format (UPF) in 2006, providing a mechanism to annotate an electric design with the power and power control intent for that design. UPF is designed to be written by a digital logic engineer to work within existing RTL formats, enabling automatic insertion of the power network and power-dependent logic, and accurate simulation of low power structures and behavior both at RTL before synthesis, and at gate level after power insertion has been completed on the backend of the design process.

In the analog universe, SPICE models have always provided a methodology for encapsulating power information. The power supply, ground, inputs and outputs for any given analog circuit are always defined.

The difficulty is that there is no single, clearly-defined approach today for mapping detailed SPICE information into the digital world, either via UPF, language extensions, or another methodology. This lack of guidance means each company's AMS verification team is creating its own solution, usually based on connecting analog supplies to binary wires in the digital side. Such methodology requires a user to simulate a "pg-netlist" version of the digital part, which is only available late in the flow, or worse, not fully aligned with validated RTL.

To help address the increased requirements around analog mixed-signal verification, Synopsys has developed the fastest cosimulation solution via a direct kernel interface between CustomSim-XA and VCS-NLP, but that only solves part of the problem. We still need a clearly defined connection between SPICE and UPF-annotated digital models that allows users to transfer power intent information from the 'real' and SNT views of UPF and VCS-NLP simulation to the 'electrical' views of SPICE.

We are actively working with industry partners to build out the complete enhanced flow for Analog Mixed-Signal power verification, but in the interim we have built out a phase one solution, which I will denote as "Real2SNT", and detail in the rest of this paper. To detail our current solution, (see Figure 1) we will take a logic block powered (in UPF) by one or more supplies that have been modeled in SPICE. First, the user will create a SV wrapper for the set of logic, extending the logic ports to the edge of the wrapper, and adding dedicated ports for the supplied. Those supply ports will allow the user to connect analog supplies using 'electrical to real' (e2r) conversion defined in the traditional CustomSim-VCS flow. A user will also create a UPF wrapper, applied to the SV wrapper, loading the original UPF (using 'load_upf –scope').

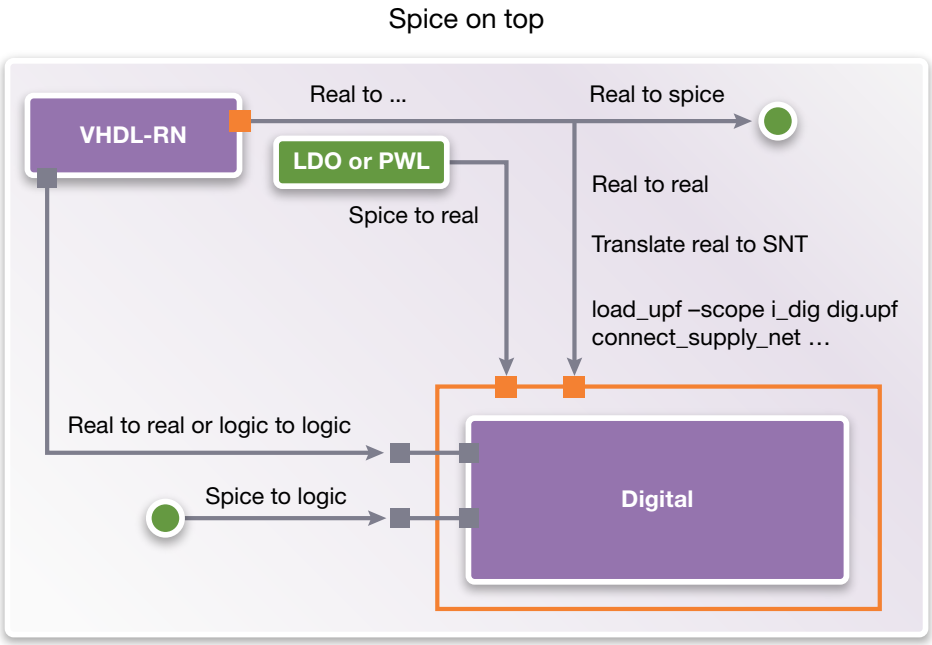
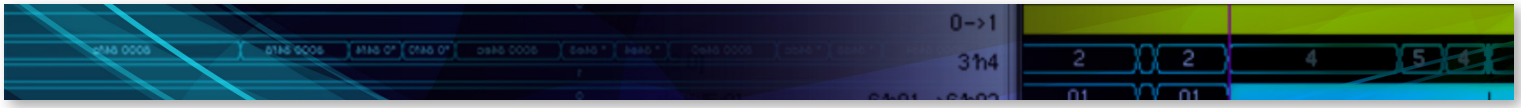


Figure 1: Transferring SPICE power simulation into a UPF format for digital verification using CustomSim-XA and VCS-NLP.

The wrapper would then connect that 'real' supply port to a supply net (defined as a SV supply net type) through a function converting the real value to the 'voltage' element of the SNT. VCS-NLP automatically mirrors the supply net from the SV wrapper to a UPF supply of the same name, mapping the Real2SNT interface from the 'electrical supply' in SPICE to a 'UPF supply' in the digital part.

Because this approach enables VCS-NLP to run with awareness of analog circuitry, users can more easily and hierarchically verify the Power State Table (PST), a key part of design specifications.

Synopsys is working with industry partners to create an automatic interface for this type of logic insertion—eliminating the wrapper and manual work that goes along with it. This specification still requires input from the community, so please contact your account team if you have requirements to contribute. We continue to work with our customers to define the best methodology for a fast, reliable link between for power verification between analog and digital verification.



Upcoming Events 2013–2014

Verification Futures Conference
http://tinyurl.com/pqrw8xe

Munich, Germany
November 18, 2013

Reading, UK
November 19, 2013

Sophia-Antipolis, France
November 21, 2013

DVCon
http://dvcon.org
San Jose, CA
March 3-6, 2014

SNUG Silicon Valley
www.synopsys.com/sv-snug
Santa Clara, CA
March 24-26, 2014

DAC
www.dac.com
San Francisco, CA
June 1-5, 2014

Resources

Functional Verification
www.synopsys.com/Functional-Verification

Debug
www.synopsys.com/Debug

Verification IP
www.synopsys.com/VIP

Hardware-Based Verification
www.synopsys.com/HW-Verification

Synopsys SolvNet
solvnet.synopsys.com

Analog Mixed Signal Verification
www.synopsys.com/AMS-Verification

FPGA-Based Prototyping
www.synopsys.com/FPGABasedPrototyping

Feedback and Submissions

We welcome your comments and suggestions. Also, tell us if you are interested in contributing to a future article. Please send your email to **avb@synopsys.com**.

SYNOPSYS®
Accelerating Innovation

Synopsys, Inc. • 700 East Middlefield Road • Mountain View, CA 94043 • www.synopsys.com

12/13.TT.CS3554/CPR/250.