

HARDWARE/SOFTWARE VERIFICATION WITH INCISIVE SOFTWARE EXTENSIONS

STEVEN BROWN, PRODUCT MANAGEMENT DIRECTOR, SYSTEM REALIZATION

CONTENTS

- Introduction 1
- Market Trends 1
- What is Needed for Effective Hardware/Software Verification 3
- Incisive Software Extensions..... 4
- Unified Hardware/Software Debugging 5
- Continuum of System Verification and Software Development Solutions... 6
- Software Execution..... 7
- System Level Low Power Verification..... 8
- Software Verification IP... 9
- Conclusion..... 9

INTRODUCTION

System bring-up and hardware/software verification are two of the most challenging and lengthy tasks in systems design, and they often threaten to delay market entry for today’s electronic devices. The complexity of the hardware/software interface has exceeded the ability of engineers to conceive and test the interactions, and to reverify when changes are introduced. This paper describes a unique functional verification automation solution that shortens the time to functionally verify software drivers along with system hardware, and helps improve the quality of the eventual system.

MARKET TRENDS

Electronic systems are increasingly differentiated by the software applications that bring core capabilities to life for the user. They perform services that enable mobile communication, decision making, information sharing, and social connections that were never possible before. The Android operating system, for example, provides a defacto standard software development environment upon which rapid application innovation is taking place. The iPhone/iPad development platform is similarly enabling innovation, and it provides unique application interoperability and integration.

Underlying these software applications are an increasing variety of devices. Each device is powered by special purpose electronics that optimize size, performance, power consumption and a variety of form factors. The hardware abstraction layer contains software drivers that uniquely enable the operating system and applications to access optimized hardware. The drivers are highly complex and customized for each device. The rapid pace of device creation, along with short product lifecycles, is shrinking schedules to create, bring-up, fully verify, and optimize the operating system running on the device.

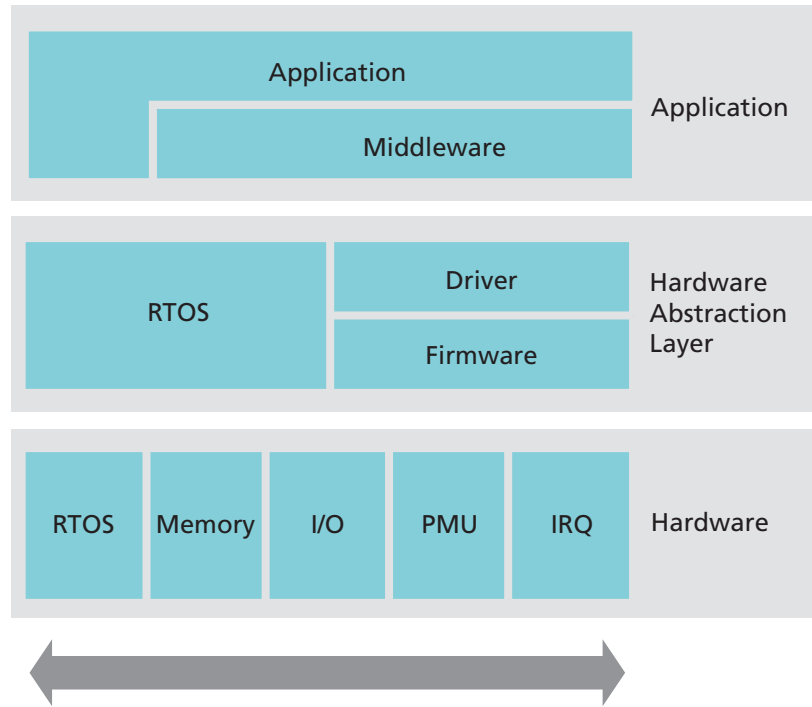


Figure 1: The software stack and view of the hardware

The trend towards application software differentiation is resulting in pressure on chip teams and suppliers to deliver software drivers along with their silicon. In the past, software teams have collaborated with separate hardware teams, but now the hardware teams are either bringing in software engineers, or simply developing the drivers themselves.

The bring-up of complex electronic systems is often quite difficult and can span weeks or several months. When the components are first combined into a system, each of the detailed design constraints are opportunities for failure. Figure 2 shows the importance of finding bugs early in a project, and the relative cost of the bug occurring in the specification of the system, the implementation processes, or the testing environment.

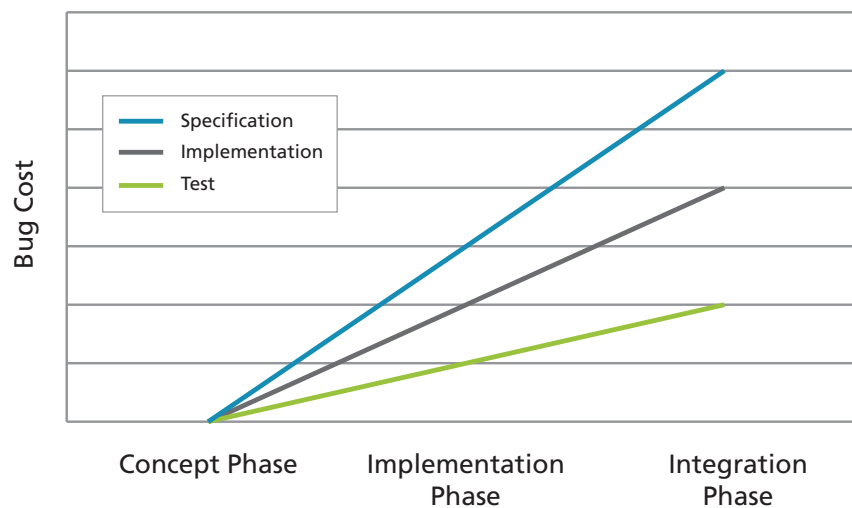


Figure 2: Relative cost of finding bugs at different stages in the project

The traditional approach to system bring-up involves encoding various system verification scenarios at each level of the system/software stack. Driving these system scenarios flushes out issues in the most common use cases, and focuses the team on the issues blocking successful bring-up. These system scenarios are often linear sequences that are driven into the system and observed at the outputs. Because embedded drivers often ship with the ASIC, the drivers must be of high quality before OS and application developers begin their work. In addition, system scenarios are becoming so complex that creating and maintaining them is burdensome, and the expected changes in the system often cause delays in the updated verification environment.

Debugging systems is quite challenging. If the team is using a prototype of the system to bring up software, debugging is often hindered by the lack of visibility into the system, and the limited ability to repeat environmental conditions to observe the bug. Teams are increasingly turning to RTL simulation and emulation for bring-up testing and debugging. These solutions require the creation of test environments that generate “interesting” system scenarios, which in turn demands engineering resources and consumes an important portion of the overall project schedule.

The relatively slower performance of these RTL solutions versus real-world hardware impedes bug fix cycles, and the late availability of the RTL often places this critical path work at the end of a project plan. Higher abstraction virtual platforms are increasingly being used to provide an early model of the system for bring-up, and they improve the speed of simulation and shorten debug turnaround. However, creating these models has been cost prohibitive for most project teams.

WHAT IS NEEDED FOR EFFECTIVE HARDWARE/SOFTWARE VERIFICATION

As ASIC teams take on software driver development and verification, they are naturally looking to extend traditional techniques for high quality functional verification to the software domain. The driver and OS teams are seeking ways to leverage automation to encode system tests and improve their maintainability. And both teams are seeking debugging solutions that make the hardware and software behavior visible to them in a unified way, yet provide accessibility to the details comfortable to engineers in each domain.

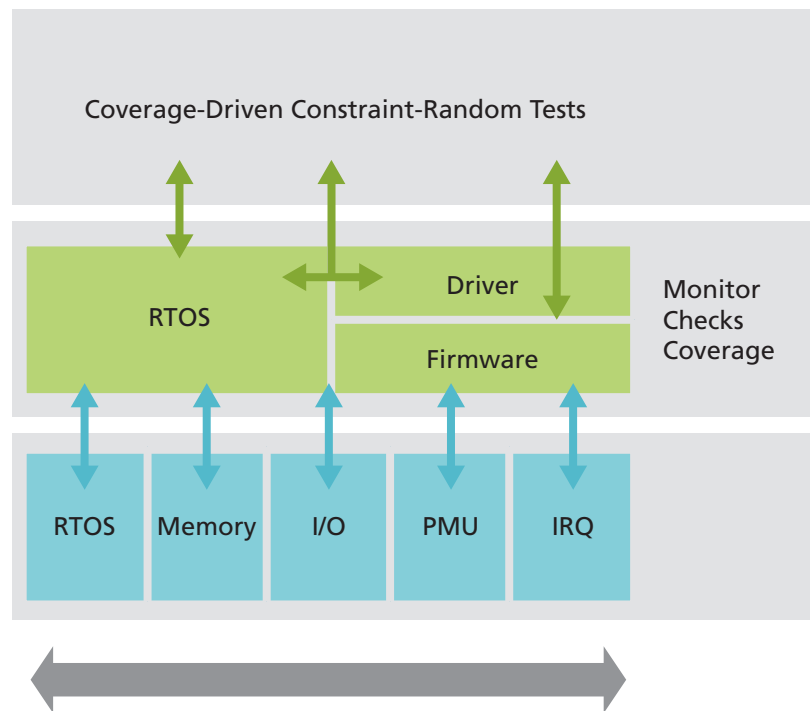


Figure 3: Applying advanced functional verification to hardware / software

Automated functional verification technology has matured in the ASIC and FPGA domain, and is now available as an industry-standard methodology called the Universal Verification Methodology (UVM). A growing number of ASIC teams are applying the UVM to the software routines of their device drivers, and to stimulate interspersed hardware activity such as interrupts and peripheral operations to test the robustness of their system.

The UVM methodology and technology provides a succinct encoding of the scenarios that are to be driven into the device for verification, allowing the tools to randomly generate various combinations of stimulus that often explore unexpected corner cases in the design, improving overall quality. The maintainability of the scenario description is significantly more efficient because of the abstract description of the behavior and leveraging of next-generation technology. The verification environment also describes expected results, and can capture the expected hardware results as well as the return values and memory results of software function calls.

Functional coverage is an important quality metric that is incorporated into the UVM verification environment description, and it complements traditional code coverage that originated from the software development domain. Both functional and code coverage are used to describe the complete set of system behaviors that must be exercised to be confident functional verification is thorough. In addition, coverage can be organized into a verification project plan, staging the measurement of functional coverage in order to address verification milestones, and organizing coverage by higher level subsystems so as to focus teams on problem areas and better manage project progress. A verification plan can be used to organize functional coverage to represent system features, milestones, and to document system quality hot spots—those areas of the design that contain several bugs due to complexity or lack of engineering focus.

Metric-driven verification is the use of a verification plan and coverage metrics to organize and manage the verification project, and optimize daily activities to reach verification closure. Executing regression suites produces a list of failing runs that typically represent bugs in the system to resolve, and coverage provides a measure of verification completion. Bugs are iteratively fixed, but the unique process of metric-driven verification is the use of coverage charts and coverage hole analysis to aid verification closure. Analyzing coverage holes provides insight into system scenarios that have not been generated, enabling the verification team to make adjustments to the verification environment to achieve more functional coverage.

This type of rigorous functional verification has not been broadly adopted for embedded software drivers, or most software projects. The complexity of drivers, systems, and the need to deliver high quality before system bring-up is necessitating the transition, and teams that have a critical time-to-market requirement are investing in this metric-driven approach.

To employ virtual platforms and enable early hardware/software integration, teams need a way to reduce the work of creating an additional model of the design. This requires that the virtual models also serve as the source for silicon design using a high-level synthesis tool and advanced functional verification. This requires a modeling methodology that provides high-performance simulation modeling for software execution, good quality of results (QoR) from the silicon synthesis process, and a verification methodology for managing the design flow.

INCISIVE SOFTWARE EXTENSIONS

The challenges of driver development and verification can be eased with Cadence® Incisive® Software Extensions. Part of the Incisive family of verification products, they enable unified hardware/software debugging and automated functional verification of hardware/software interactions.

Incisive Software Extensions use an executable verification plan (vPlan) to organize and apply automated functional verification and speed the time to uncover hidden bugs in the system, shortening the overall verification project schedule. Automation is used to generate system scenarios to exercise the hardware and software portions of the design, to check results for bad

behavior, and measure functional verification completion. This reduces the effort to explore the design, uncovers complex bugs earlier in the project schedule, and enables teams to prioritize the bugs the team addresses to optimize their efficiency.

Incisive Software Extensions provide unified hardware/software debugging tools to shorten the time to identify, characterize, and resolve system bring-up issues. Hardware and software domains are best debugged with different information paradigms and debugging approaches, specifically those that match the training and design approaches for each team. A unified debug environment provides all of the techniques and data visualization designers would expect for each domain, and connects these features in the underlying database and GUI operations. It is this unique combined environment that shortens the time to find and resolve bugs.

Incisive Software Extensions can be used to verify the functionality of the software. By applying the UVM to randomly execute the function calls in each software package, the quality of software can be greatly improved compared to previous, less rigorous testing strategies. This approach to verification of software is gaining attention as software packages are increasingly reused in new systems.

The modular approach of the UVM enables the separation of software into its naturally separate units for verification. Figure 4 shows how Incisive Software Extensions are applied to each layer in the system, and the interaction of each layer.

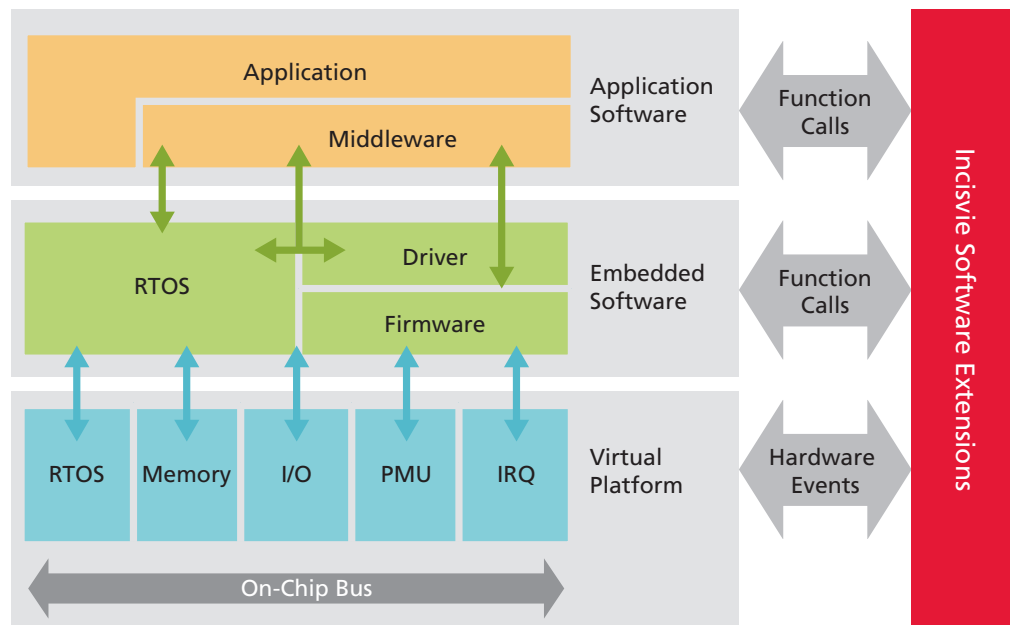


Figure 4: Applying the UVM to application and embedded software and hardware

UNIFIED HARDWARE/SOFTWARE DEBUGGING

Incisive Software Extensions provide a unique hardware/software debugging solution, combining details from RTL or SystemC® debugging capabilities with traditional software debugging. Each domain has been specialized for the concepts and preferences of associated engineers, resulting in very unique styles and different technical solutions. By unifying these into a single database, single GUI, and single run control mechanism, the debug experience for the user is streamlined. Users are able to delve into the domain that is least familiar to them, in an environment where the information and interactions are tightly coupled with that which they are familiar.

This also aids in team debugging, when a system bug isn't obviously in either hardware or software, but is somewhere between them, or resident in both domains. Engineers from each specialization are able to deliver verification environments from their own debugging to their team mates, enabling them to pick up where they left off. Or they can easily work together and debug simultaneously.

Embedded Software Trace

Embedded Software Trace integrates embedded software windows into SimVision debug windows, the graphical user display for the Incisive family of products. It provides source views of the embedded software under execution. The Incisive Software Extensions connection to the processor model provides an accurate program counter, enables single stepping, allows breakpoints to be set and triggered, and permits software variables to be plotted alongside RTL or SystemC hardware signals.

CONTINUUM OF SYSTEM VERIFICATION AND SOFTWARE DEVELOPMENT SOLUTIONS

A significant benefit of using Incisive Software Extensions is the capability to connect to a spectrum of design virtualization solutions. Each provides a unique performance, accuracy, and availability/cost tradeoff. The connection of Incisive Software Extensions to each of these platforms presents a common approach to verification and debugging, and leverages the investment in the system verification environment for more applications.

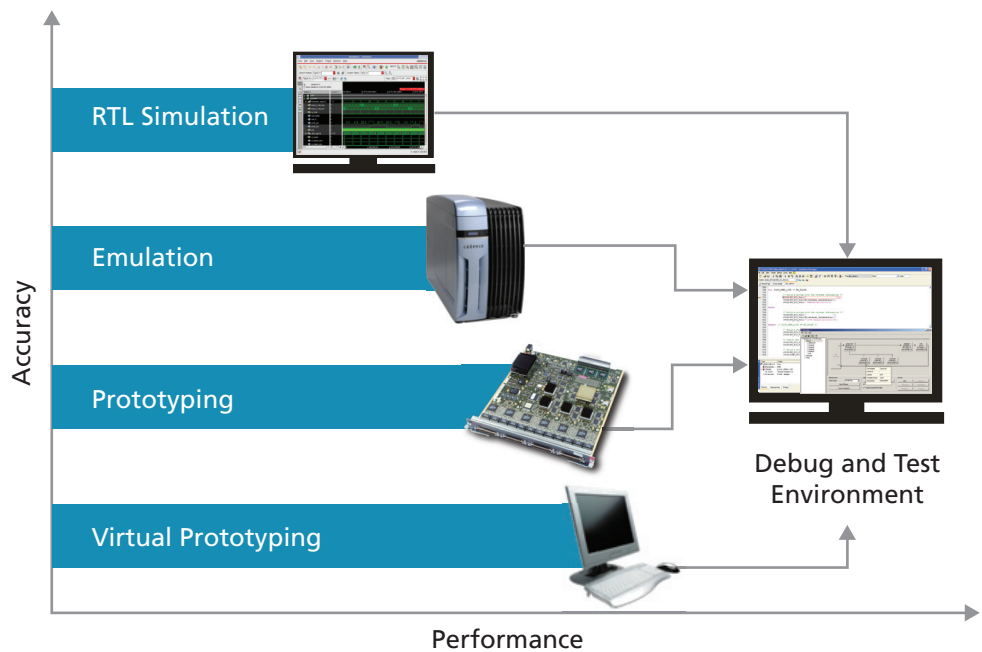


Figure 5: Virtualization spectrum performance vs. accuracy

The Cadence Incisive SystemC simulator can be used to execute SystemC-based virtual platforms with Incisive Software Extensions. This provides a mechanism to perform advanced functional verification of software and hardware early in the development of both, finding major bugs earlier, and enabling earlier debugging when the design description is simpler and turnaround time is very short.

The SystemC code may model the behavior of the design without timing accuracy, providing the highest performance of transaction level behavior. This modeling approach is useful for major functionality, but does not support verification of timing-dependent system behavior.

When following the TLM-driven design and verification methodology¹, these SystemC models can be synthesized to RTL using the Cadence C-to-Silicon Compiler, optimizing for the desired area, timing, and power constraints.

Incisive RTL simulation executes the detailed micro-architecture of the hardware and provides the accurate timing behavior of the design for verification with software. This is important for timing-dependent software operations such as driver interactions with hardware.

RTL simulation is often too slow for software execution due to the size of the design and the detail being simulated. Cadence Palladium® XP Verification Computing Platform acceleration provides visibility and controllability of the design, and MHz execution, which is reasonable for software drivers and limited verification of higher layers in the software stack. This speed advantage can be critical for very long Incisive Software Extensions execution runs.

FPGAs or silicon prototypes can be used with Incisive Software Extensions, and the user can apply its verification environment and measure coverage. While there is limited visibility into these devices, the consistent verification environment allows easy replay of error cases using Palladium or Incisive for debugging.

SOFTWARE EXECUTION

Execution of software is achieved with a model of the processor that is connected to the design under test. Incisive Software Extensions allow any form of processor model to be used to execute the software instruction stream, enabling the user to choose between various available approaches so as to trade off speed, accuracy, and cost.

Each processor model requires a basic setup to connect to the verification environment to fetch instructions, control instruction execution, and capture results into the design. Setup procedures include connecting to the program counter to enable single stepping, setting breakpoints, and probing variable values during simulation.

An instruction set simulator (ISS) emulates the register and program counter behavior of the processor by abstracting the internal state machine of the processor itself. Therefore, the performance is often the fastest for simulation, but the ISS approach lacks the accuracy to perform system timing/performance analysis. The current generation of ISS technology employs a just-in-time (JIT) translation of each instruction into the instruction language of the workstation executing the simulation. While this approach can achieve hundreds of MHz performance, it requires a sophisticated approach to handle today's multi-core and speculative execution processor architectures. Examples of current generation ISS solutions are the ARM Fast Models and Imperas Open Virtual Platforms (OVP) technologies.

Host code execution is the act of cross-compiling the software to execute on the workstation used for simulation. Incisive Software Extensions regressions run quickly using host code execution, enabling significant daily regression completion on a workstation farm.

An RTL model of the processor is the most accurate for timing and architectural execution. There are certain system conditions, analyses, and debugging that require this level of accuracy. Unfortunately this is also one of the slowest forms of instruction execution. Adding the signal activity of the processor to an already heavy SoC simulation makes execution and turnaround time lengthy.

An accelerator like the Palladium system can be used to speed up the RTL execution of the processor and the design by orders of magnitude, and continue to provide the debug visibility and controllability of the design along with the testbench automation of Incisive Software Extensions

The TLM-driven design and verification methodology describes code architecture for virtual platform development, and for optimizing high-level synthesis and functional verification. The book "TLM-Driven Design and Verification Methodology" by Brian Bailey, et. al. provides an overview.

A processor model can be used in its fabricated silicon form, either in an FPGA, or in a special purpose package. By connecting this silicon on a PCB and connecting the board to the workstation, the instructions and execution results can be attached to the design simulation and provide both high accuracy and high performance benefits.

Each of these approaches to modeling the design and processor enable different verification benefits. Some are available early in the project to allow early software development and bug discovery. Some provide the detailed accuracy needed to finish verification. The advantage of Incisive Software Extensions is the ability to use one verification strategy and environment to drive all of these points on the project schedule and accuracy/performance spectrum. There are several combinations of design and processor execution that can be combined with Incisive Software Extensions. This flexibility gives teams multiple options for performing hardware/software verification at multiple stages in the design flow.

		Processor Model				
		ISS	Host Code	Incisive RTL	Palladium RTL	Silicon
Design	Incisive SystemC	Performance & Schedule	Performance & Schedule	Accuracy of Processor	Accuracy of Processor	Accuracy & Performance
	Incisive RTL	Accuracy of Design	Accuracy of Design	System Accuracy	Accuracy & Performance	Accuracy & Performance
	Palladium RTL	Accuracy & Performance	Accuracy & Performance	Accuracy & Performance	Accuracy & Performance	Accuracy & Performance
	FPGA	Performance	Performance	Accuracy	Accuracy & Performance	Accuracy & Performance

Figure 6: Incisive Software Extensions verification goals and modeling strategies

By charting the options for modeling the processor and the design, and the execution strategy for each, a team can create a comprehensive hardware/software verification strategy made more efficient by the reuse of Incisive Software Extensions.

SYSTEM-LEVEL LOW-POWER VERIFICATION

Power is a critical system design criteria, and increasingly software has significant influence over power consumption. The many power management schemes create functional behavior complexities that need to be verified. Hardware can be designed with many advanced low-power features, but if the software fails to take advantage of these then the application will not see any low-power benefits.

Incisive Software Extensions uniquely address this complexity by enabling the team to specify the IP level, SoC level, and software level of power management functionality. Hardware has specific clock gating and power control module functionality that must be verified. Power modes are turned on and off via the embedded software drivers. Application software provides the system user mode context that implies desired power optimization behavior as a tradeoff against performance and function not actively required.

The vPlan can be used to define the complete functional verification for all power modes and scenarios, and the features available in each mode. It instruments the power modes and sequences, the behavior during each mode, the power control logic, and the interfaces between power domains. This enables a team to track the verification of all power mode combinations to ensure high quality.

SOFTWARE VERIFICATION IP

Setting up the verification environment is semi-automated, generating the verification code directly from the embedded software. It generates the verification environment to call each of the software functions and to randomize the calls as well as the data variables.

Incisive Software Extensions uses verification environments that are written in the *e* and SystemVerilog high-level verification languages (HVLs). By following the Universal Verification Methodology (UVM), verification code can be easily reused in subsequent projects that reuse any of the hardware or software components.

CONCLUSION

Incisive Software Extensions offer a unique combination of hardware/software functional verification across a spectrum of virtualization platforms, and unified hardware/software debugging. The advanced functional verification enables early discovery and resolution of difficult hardware/software bugs using highly productive automation. The spectrum of virtualization platforms supporting Incisive Software Extensions reduces the effort cost and time for maximizing the leverage of each platform to produce high-quality systems in the shortest schedules. And the unified hardware/software debugging shortens the time for teams to collaboratively uncover difficult hardware/software bugs.

For more information
contact Cadence sales at:
+1.408.943.1234
or log on to:
**[www.cadence.com/
contact_us](http://www.cadence.com/contact_us)**

cadence[®]

Cadence Design Systems, Inc.

CORPORATE HEADQUARTERS

2655 Seely Avenue
San Jose, CA 95134
P: +1.800.746.6223 (*within US*)
+1.408.943.1234 (*outside US*)
F: +1.408.943.5001
www.cadence.com