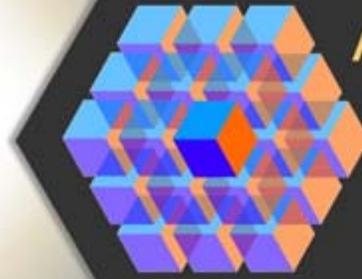


sponsored by

Gabe on EDA • EDAMarket



# Assembling the Future

A Newsletter About the Design  
and Production of Electronics

ISSUE 022 • JANUARY 2013

[SUBSCRIBE](#)

[PDF and Archives](#)

[twitter](#)

## In this issue:

- [Prototyping Is Both Advanced and Expensive](#)  
by Gabe Moretti
- [Accelerate Software Development with High-Performance FPGA-based Prototyping](#)  
by Neil Songcuan
- [Defining the Various Use Models for System Prototyping](#)  
by Frank Schirrmeister
- [Accuracy, Now More Than Ever](#)  
by Bill Neifert
- [System Prototyping and Verification Reuse](#)  
by Thomas L. Anderson

---

## Prototyping Is Both Advanced and Expensive

Gabe Moretti

As the four articles in this issue document, System Level Prototyping has come a long way, not just from the early days of vendor specific prototyping boards, but from a decade ago. The question is where do we go from here? I believe the short term answer has two components. We must be to extend the definition of system to the entire environment in which the electronic system will operate. And we must provide better hardware/firmware tradeoff tools.

Proper analysis of the projected product should include the die package, the board on which it sits,

the enclosure in which it operates, and even the nearby environment the product will operate in. By nearby environment I mean the hand of the operator as in the case of a phone, the engine compartment of a ECU, the rack of a router.

When necessary the industry should develop appropriate standard semantics and syntax to transfer the information about the physical characteristics of a functioning electronic system to analysis tools developed by other engineering disciplines. We know that the most important characteristics to be output are: power consumption, thermal characteristics, and magnetic field effects.

Power consumption is important in order to determine the characteristics of the power supply required. The ability to deliver the appropriate current determines the size and life of a battery in a handheld device, and allows engineers to calculate the total draw from a distribution network for fixed devices.

The size of the battery is critical in handheld devices because it contributes to the form factor of the product as well as the amount of operational time between charges. Clearly this characteristic can be accurately measured once the IC is finished and is in its ultimate operating environment. But it would be nice to be able to know this as soon as possible so that the other parts of the product can be designed and developed with the knowledge of the operating conditions in the field.

This last consideration also applies to thermal conditions. The heat generated by the electronic system spreads to the rest of the product and must be dissipated into the surrounding environment. At the architectural level there exist the possibility of trading off between application specific hardware and firmware. Executing software instructions on a processor generates more heat than executing the same function in hardware. But using firmware assures more flexibility in generating members of a family of products in a less expensive manner. At least this is the general wisdom. I have come to doubt that this is such a general situation.

Thermal modeling is becoming more popular and good models and analysis tools exist for a completed IC. Once again it would be helpful to be able to model the system at the architectural stage using realistic thermal values for the IC.

Electromagnetic radiation is the third important characteristic of an IC when considered as an executing part of the total system. How far should an IC be situated in a smart phone from the antenna? Is there any shielding required? What is the impact on the IC on the immediate electronic surrounding? These are all questions that a product architect needs to answer as soon as possible in order to avoid costly redesign during the development process. We need to find a way to produce reliable models of an electronic system at the architectural level that allow engineers to place the components in the total product layout in such a way to achieve maximum reliability.

#### Increased Use of Hardware

As the articles in this issue point out, software verification and debug is difficult and thus expensive. I believe that implementing the same function in hardware could reduce development time significantly enough to make a difference in the profitability of a product. As the availability of proven IP blocks grows, and with it verification standards, hardware development can require less verification effort due to "correct by construction" techniques. Together with flexibility, there have been two of the major advantages of using firmware in IC design: flexibility of derivative products

and space limitations.

Intelligent design of a product can improve the opportunity to produce derivative products reliably and inexpensively using hardware modules. Given an accurate catalogue of available IP cores, an architect can generate a number of alternative designs that cover the entire product family offering a range of functionality.

The number of available transistors in a die is such that adding hardware blocks is almost never an issue from a real estate point of view. Experience with firmware development has demonstrated that developing, debugging, and prototyping firmware is expensive, both from a time, manpower, and tools cost point of view. The EDA industry has addressed the challenge, it is obvious from the articles in the issue, but the demand for support of firmware development continues to increase, just as processors' complexity grows.

The alternative is a growing Software IP industry. Given development costs, such an industry is at the point of becoming viable as a business for EDA companies. Our industry should deal with all of the components of an IC. System Level tools have finally freed us from using a four letter word (EHDA) to describe ourselves.

---

## Accelerate Software Development with High-Performance FPGA-based Prototyping

Neil Songcuan, Product Marketing Manager, Synopsys

Many design teams enjoy the benefits of using FPGAs for SoC prototyping. Growing pressure on chip delivery schedules and the economic penalties of being late to market or delivering a defective product are driving design teams to deploy FPGA-based prototyping more broadly. FPGA-based prototyping provides the ability to enable parallel hardware-software development, allowing system integration to begin months before SoC silicon is available, has encouraged design teams to use it for software development, hardware/software integration and system validation.

As businesses deploy FPGA-based prototyping more widely across their design teams, they are beginning to ask for more from their development environments. In the past, many design teams developed their own prototyping boards in-house, often tailoring them toward the needs of each specific project. Today, reusability is a growing requirement, enabling the use of a common FPGA-based prototyping environment for different designs. Today, chip companies distribute their engineering resources globally, so having a common prototyping environment improves design team productivity. In addition, chip developers are constantly looking for ways to make the most efficient use of their engineers, and creating sophisticated SoC prototypes is often not deemed a core focus area.

These trends are compelling design teams to look for alternatives to their in-house prototyping

environments.

## HAPS 70: Sixth-Generation FPGA-based Prototyping System

Synopsys has developed its sixth-generation HAPS-70 series system to be more flexible and reusable than in-house or other commercially-available prototyping systems. The HAPS-70 systems take advantage of a scalable architecture and the latest generation Xilinx Virtex-7 2000T FPGAs to support a wide range of design sizes with capacities from 12 to 144 million ASIC gates. Synopsys engineers have focused on building a robust environment that reduces the risks for design teams looking to adopt and use FPGA-based prototyping across a range of projects. Key HAPS-70 features include:

- Modular system architecture scales from 12 to 144 million ASIC gates to accommodate a range of design sizes, from individual IP blocks to processor sub-systems to complete SoCs
- Enhanced HapsTrak 3 I/O connector technology with high speed time-domain multiplexing delivers up to 3x performance improvement in data throughput over traditional pin multiplexing
- System definition and bring-up utilities speed hardware assembly and ensure the prototype's electro-mechanical integrity
- Advanced power and cooling management
- Improve debug efficiency with 100x greater visibility and 8x faster download time of debug trace buffer data
- Advanced use modes including co-simulation, transaction-based verification, and hybrid prototyping

Providing a high-quality, reusable prototyping environment requires more than just hardware; Synopsys has enhanced its FPGA-based prototyping software tools – including its Certify® multi-FPGA prototyping environment and Identify® integrated debug tools – to take advantage of the new HAPS-70 hardware features.

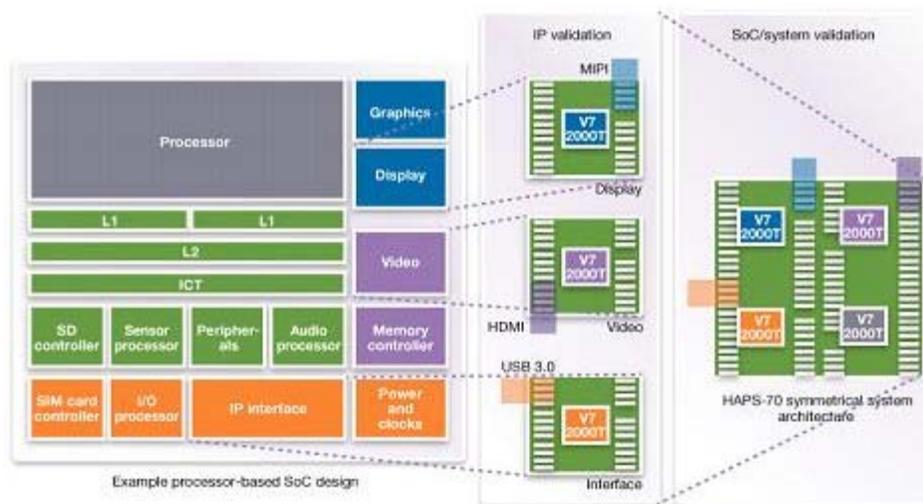
### Modular Symmetrical System Architecture

Synopsys has designed the HAPS-70 prototyping solution around a new symmetrical system architecture (SSA).

SSA defines the mechanical symmetry of the PCB and connector layout of the system, which enables design teams to use the HAPS-70 system in a modular way. The architecture ensures pin-to-pin forward compatibility for constraints, cables and daughter boards, which facilitates easy movement of the physical connections and complete prototyping designs. The architecture also specifies the use of connectors with delay-matched characteristics so that designers can move components with minimum delay to timing, which makes it easier to synchronize clocks across multiple FPGAs when scaling to larger systems.

The scalable architecture is key to providing flexible support for complex IP blocks and SoC designs alike. For example, design teams can take smaller designs targeting smaller HAPS systems and easily migrate them to larger HAPS systems that represent the entire SoC.

SSA ensures that design teams are no longer constrained to using just one, two or four FPGAs – they can connect multiple FPGAs to create large capacity systems –up to 144 million ASIC gates using 12 FPGAs. SSA enables design teams to take a bottom-up, ASIC-like approach to developing prototypes; reusing their subsystem and IP prototypes and ultimately saving system-on-chip bring-up time. For example, Figure 1 shows how HAPS-70 enables a design team to validate three separate subsystems – MIPI display, HDMI video and USB 3.0 – before combining them with the processor in the top-level configuration.



[ Figure 1: HAPS-70 symmetrical system architecture enables scalable prototyping ]

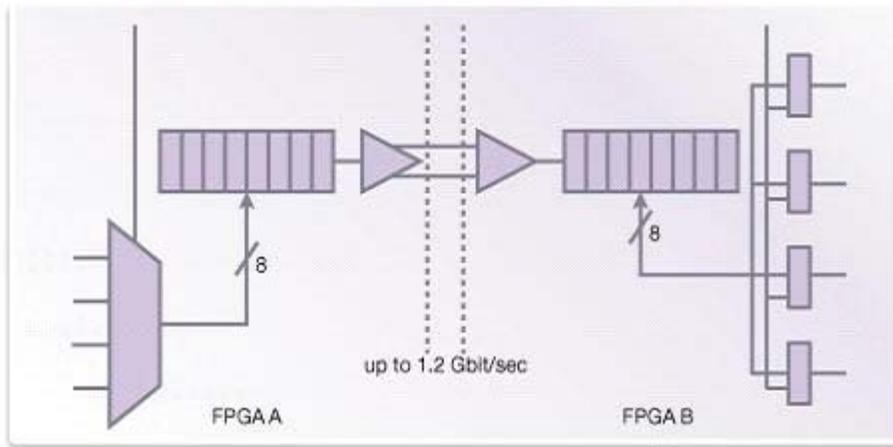
Synopsys Certify multi-FPGA design environment with HAPS system awareness supports and automates the hierarchical project flow, resulting in faster FPGA-based prototype bring-up.

#### Automated High-Speed Time-Domain Multiplexing

Even with today's high-capacity FPGAs, design teams frequently have to partition their largest SoC designs across multiple FPGAs in order to fit them into the prototyping system. The process of partitioning the chip creates another problem; passing sometimes thousands of signals between FPGAs.

Using pin multiplexing can be an effective way of squeezing multiple signals through a single off-chip connector. However, while solving one problem, pin multiplexing creates another –performance bottlenecks emerge as thousands of signals pass from one FPGA device to another.

The HAPS-70 system supports the automated insertion of high-speed time-domain multiplexing (HSTDM). HSTDM is a capability that is unique to the HAPS-70 FPGA-based prototyping system. HAPS' Certify software packs up signals and transmits them between FPGAs at over 1 Gbit/s. HSTDM is typically 3x faster than traditional pin-multiplexing solutions. Prototyping systems using HSTDM can run at near real-time speeds and handle data from real-world I/O.



[ Figure 2: HAPS-70 automates HSTDM ]

HSTDM demonstrates the importance of tightly integrating hardware and software for the prototyping system, where Certify takes advantage of its knowledge of detailed timing information about the HAPS-70 system, connectors and cables, and enables customizations and optimizations not available with home grown hardware prototypes.



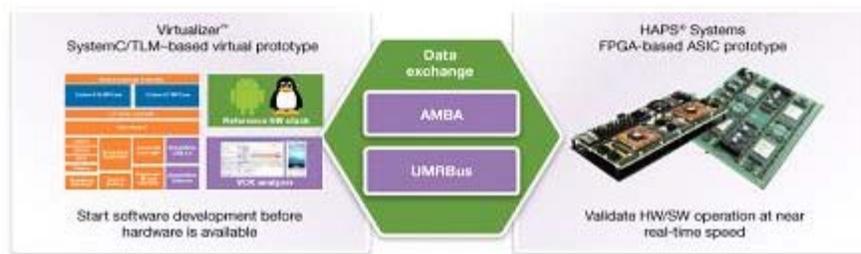
[ Figure 3: Enhanced HapsTrak 3 connector technology supports high-speed I/O interfaces ]

#### High-Speed UMRBus Interface

The capabilities in HAPS prototypes extend their usefulness across the entire product development organization, including the ASIC, software and systems development teams. And, HAPS integrates well into the broader design flow.

The enhanced HAPS-70 Universal Multi-Resource Bus (UMRBus) supports high-speed connectivity – up to 400 MB/s – between the HAPS prototyping environment and a host workstation. This high-performance, low-latency link enables the creation of hybrid prototypes, thanks to the efficient exchange of data between the HAPS system and a SystemC/TLM-based virtual prototype such as those based on Synopsys' Virtualizer™ tool set, which runs on the host (Figure 4). Using this kind of

configuration, design teams can take advantage of advanced use modes, such as co-simulating hierarchical blocks using VCS® and other simulators as well as earlier software development with virtual prototypes. This approach allows the use of non-synthesizable verification IP, which can run on the host platform and be used to exercise the FPGA-based prototype.



[ Figure 4: Hybrid prototyping solution ]

High-speed connectivity to the host workstation is key to enabling remote access to the physical prototyping system, allowing remotely located designers to easily access registers and manipulate the HAPS-70 prototypes across their corporate network.

#### Easy-to-Use Integrated Software Flow

To prototype an ASIC design using FPGAs, the design team must convert certain design elements to structures that FPGA implementation tools can recognize. These elements, such as ASIC gate-level components or gated-clock tree structures, can be difficult and time-consuming to convert manually. The Certify software automatically recognizes and converts these ASIC-specific constructs into equivalent FPGA structures.

Certify enables automated (and manual) partitioning to support multi-FPGA implementation, to provide efficient conversion and partitioning of large ASIC designs onto HAPS multi-FPGA prototyping boards using a graphical user interface (GUI) flow guide.

Using the [Synplify® Premier](#) synthesis engine, Certify automatically maps ASIC-style RTL source code and DesignWare® IP to multiple FPGAs, providing immediate FPGA resource consumption feedback during the partitioning process.

The Certify software is tightly integrated into Synopsys' FPGA-based prototyping hardware and software flow. Board descriptions for HAPS systems are built into the Certify tool, allowing for immediate productivity with almost no setup time. Certify software assures optimal performance because it automatically takes advantage of HAPS signals to provide HSTDM, which ensures the fastest available connections between FPGAs.

#### Advanced Debug

Gone are the days of using logic analyzers to debug FPGA hardware. Now FPGA vendors provide embedded probes to view activity at the post-synthesis gate-level. Multi-FPGA prototypes make the debug problem harder because design elements can be split across devices following implementation at the gate level. Design teams can be more productive if they are able to drive debug from the comfort of their familiar RTL.

The HAPS-70 system integrates with Identify software to support multiple debug capabilities that help designers locate bugs quickly and efficiently. The HAPS Aware Identify debug features provide simulator-like visibility of the prototyping project at the RTL source level. Designers can take advantage of multi-FPGA debug visibility to debug their prototypes regardless of the design partitioning and benefit from simulator-like signal browsing.

Identify's Deep Trace Debug enhanced visibility gives users as much as 100x additional design visibility by utilizing off-chip, external memory storage. Sample results are available quickly over the high-performance HAPS UMRBus Interface Pod link between the debugger workstation and the HAPS-70 system.

Identify also supports the following advanced debug features:

- Ability to instrument and debug an advanced FPGA design directly from RTL source code
- Advanced trigger creation allows the viewing of desired design operation scenarios and the ability to probe specific nodes in the circuit
- Visibility into the internal design while operating at full speed
- Display of debug results superimposed on top of RTL source, RTL structural view, or with a waveform viewer
- Instrument an FPGA-based ASIC prototype prior to device partitioning and planning

## Summary

Synopsys' HAPS-70 Series prototyping systems help design teams maximize their design productivity by providing a high-performance, scalable FPGA-based prototyping solution that can be used across multiple projects and engineering locations. The HAPS-70's symmetrical system architecture enables designers to create prototypes of up to 144 million ASIC gates, using a modular approach that supports hierarchical assembly of high-capacity prototypes from their constituent subsystems.

Automated HSTDM yields a 3x higher performance compared to traditional pin-multiplexing methods, which increases interconnect bandwidth and enables HAPS-70 based systems to run at near real-world speeds using data from near real-world I/O.

The hardware and software features that combine to make up the HAPS-70 prototyping system enable design teams to create scalable, high-performance and reusable prototypes for use earlier in the design cycle. Deploying FPGA-based prototypes for earlier software development, hardware/software integration and system validation is helping design teams build next generation designs faster, with less risk and more profitability.

## About The Author

Neil Songcuan is a senior product marketing manager, who is responsible for the FPGA-based Prototyping Solution at Synopsys. His experience includes the areas of semiconductor, hardware-

assisted verification and system validation. Neil has held various marketing management positions with Synplicity, Mentor Graphics and IKOS Systems. Additionally, Neil worked in customer marketing and application engineering roles with Altera Corporation. Neil holds a B.S. degree in Electrical Engineering from San Jose State University.

---

## Defining the Various Use Models for System Prototyping

Frank Schirrmeister, senior director, product marketing, Cadence Design Systems

In EDA we use the terms "system" and "prototyping" in various different situations with a clear lack of discipline and consistency, so it is worthwhile to try brief definitions first. According to the Merriam-Webster dictionary, a "prototype" is "an object that exhibits the essential features of a later type" or "a first full-scale and usually functional form of a new type or design of a construction." A "system" is defined as "a regularly interacting or interdependent group of items forming a unified whole."

Well, combining the two definitions one can describe a lot of different processes, or intermediate results of our EDA tools, as "System Prototypes." Let's wade through the chaos and try to add some structure.

So first, what are the underlying drivers requiring systems to be prototyped? From a semiconductor perspective, further miniaturization towards smaller technology nodes has led to constant growth in design complexity. For a while in the semiconductor industry, a decrease in overall ASIC design starts led to increased pressure to reach acceptable returns on investment, and required first pass success to ensure proper monetization. Today, design starts are expected to show a modest growth starting in 2015. However, the need to get proper ROIs is not going away at all, given the ever-rising NRE cost of semiconductor design.

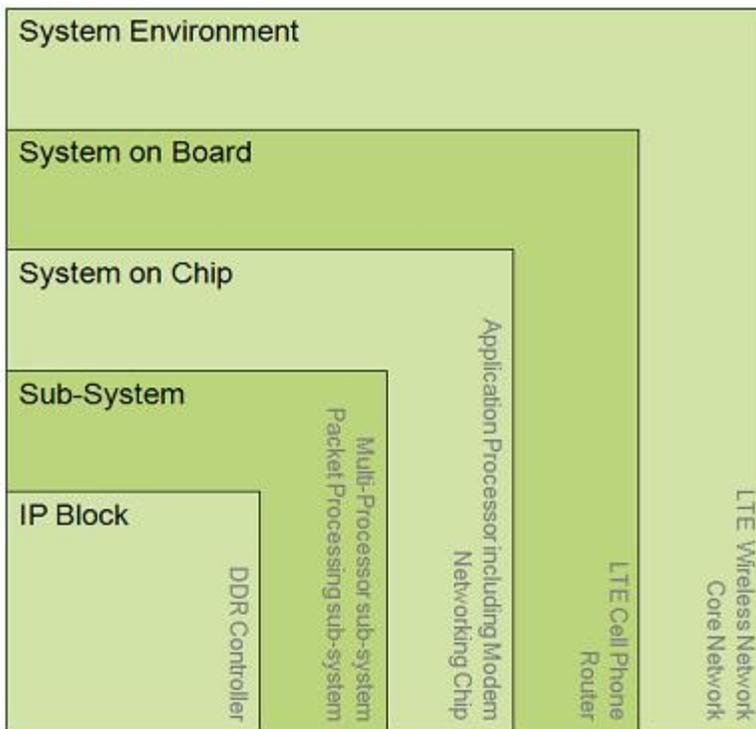
Next, on-chip programmability with a strong trend towards multicore architecture, combined with a rapid increase of embedded software content, leads to a need to prototype hardware for the purpose of software development. It also requires design teams to prototype the combination of hardware and software to make sure the integrated system works correctly. Intellectual property (IP) reuse in hardware is exceeding 60% of the semiconductor content and growing fast. In addition IP reuse of software is growing at very fast speeds, both as software IP accompanying hardware IP in the form of drivers, as well as IP in form of operating systems (OS) forming an abstraction layer between hardware and software applications.

Moreover, application specificity is driving the need towards basic hardware-software platforms for application domains like wireless, networking and automotive. This specificity is combined with stringent low-power requirements that drive the need to offer more and more functionality to users within constantly shrinking power envelopes.

Finally, an increase in the analog/mixed-signal portion of semiconductor and system designs is adding the need to make sure that the digital designs interact appropriately with the analog environment.

Going back to the Merriam-Webster definition of a prototype, the notion of "exhibiting the essential features of a later type" depends on the intended use model, of which there are four basic approaches. First, there is pure verification of hardware. Depending on the verification task, this requires various levels of depth of insight into the hardware. Second, there is verification of software, sometimes requiring access to the interface of the hardware on which it executes. Third, there is verification of hardware and software in combination, often requiring access to both hardware and software details. Finally there is validation of parameters like performance, power, and resource usage, confirming that implementation decisions were correct and meet the design intent.

An orthogonal concern is the issue of scope, defined by what the "system" actually is from the designer's perspective. A "regularly interacting or interdependent group of items forming a unified whole" leaves up to interpretation the complexity of the "item" itself. From an electronic design perspective an "item" can be a hardware block like a DDR memory interface or processor, that is part of a hardware/software sub-system for audio/video processing, that is part of a system on chip called the application processor for a mobile internet device (MID), that is part of a system of chips comprising the MID itself including analog/mixed-signal and electro-mechanical aspects, that again is part of an even bigger system, for example the network it resides in. The definition of "system" is a simple matter of perspective. As indicated in Figure 1, this is true for various application domains, like in mobile wireless or networking.



[ Figure 1 – Different Level of System Scope in Networking and Mobile Wireless ]

Combining the last two issues – use model and scope – allows us to better understand prototyping

requirements. Verifying software at the sub-system level would mean the verification of the various audio and video codecs. At the scope of the application processor these become "items" in themselves, and verification will focus on the OS experience, assuming the audio/video codecs underneath work as expected and are defect free.

At the MID level the focus of software verification will be the user experience of using applications on the device and at the network level the verification will focus on the interaction with the system environment – will my application download correctly and access databases on the network appropriately? Similar considerations apply for hardware verification, hardware/software verification (where they interact) and performance verification – which is often coupled to subjective elements like "is it fast enough" or "does the battery last long enough."

The point of prototyping a system is that when it is installed it needs to work correctly. The complexity of the system determines on how it can be prototyped. To illustrate the complexity effects, at the CDNLive India conference in 2012, ARM presented its verification approach for the ARM big.LITTLE sub-system. In this approach, ARM identifies five separate design and verification phases - "specification/planning", "implementation" leading to alpha release, "testing/debug" leading to beta release, "coverage closure" leading to limited availability customer release, and then a "stress testing" phase leading to the actual product release.

In the "implementation" phase, RTL simulation at about 100Hz is the primary engine and test benches are developed and brought up for the levels "unit" and "top". In the "testing/debug" phase RTL simulation is complemented by emulation running at MHz speeds once RTL maturity allows it. Unit-level testing/debug and top-level directed testing/debug are complemented with system-level test bench integration and bring-up. In the coverage closure phase for the unit-level, as well as the top-level, tens of billions of cycles are performed per week. In parallel, system-level software testing and debug runs in emulation at about a trillion cycles per week.

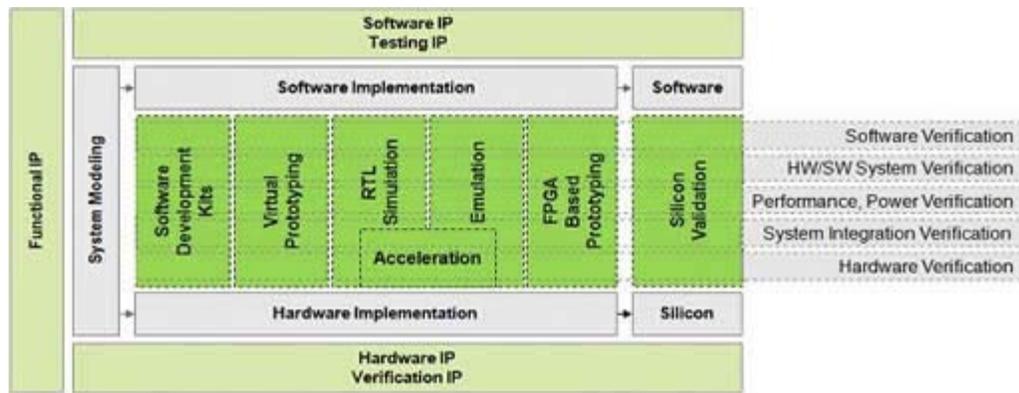
In the "stress testing" phase, before general release to customers, unit level requires 100's of billions of cycles per week, as does the top-level random soak testing. At the system-level, emulation is continued at a trillion cycles per week, and once RTL is brought up in FPGA prototypes they run at a quadrillion cycles per week. Finally, once test silicon is available, silicon stress testing runs at about 1 GHz with tens of quadrillion cycles per week.

This makes it painfully clear that verification in itself is an unbounded problem -- it is never complete. As a result, in the area of system prototyping, two basic vectors are followed – verification is done either faster, smarter or both.

The different engines EDA vendors provide naturally allow a progression of faster execution. Pure RTL simulation runs in the range of Hz. Acceleration, commonly referred to as the combination of hardware assisted verification and RTL simulation, can bring a 100x to 1000x speedup. Moving the design completely into the hardware – in-circuit emulation – allows execution in the MHz range. When more mature RTL is mapped into FPGA based prototypes (requiring bigger effort due to the required RTL changes), execution speed of tens of MHz can be achieved. All these engines execute the actual RTL at different speed levels. Similar situations are true at the gate level, for which different engines execute at different speeds.

Figure 2 illustrates these different engines with an overlay of the various verification tasks.

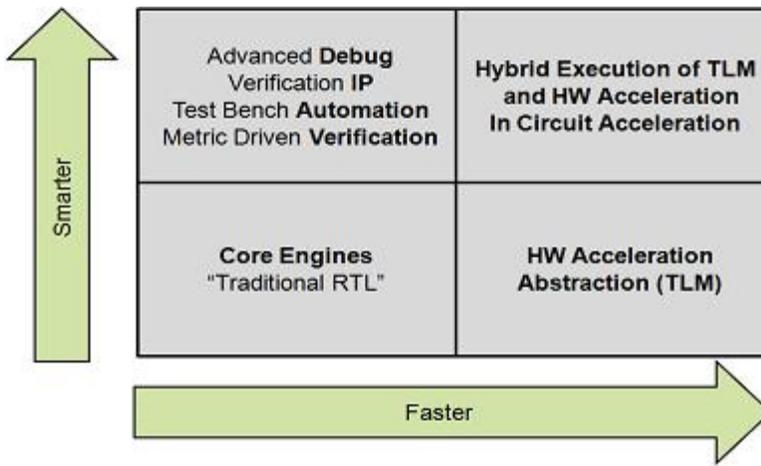
Depending on the scope of the system these tasks can be performed on any engine.



[ Figure 2 – Different execution engines for hardware/software execution with an overlay of verification tasks. ]

In addition to modulating the speed itself, verification is done "smarter" as well. The focus here is not on the fastest execution itself, but rather on abstracting the verification task appropriately, or focusing on only the important cycles to be executed to answer a specific verification question. With abstraction, just as RTL is an abstraction to gate-level simulation, transaction-level simulation is an abstraction to RTL simulation. Today, this is where functionally register accurate virtual prototypes of the hardware allow execution of the actual software. In the area of smarter verification, techniques like directed testing and choosing the right verification patterns have found adoption. Their aim is to exactly test the right corner cases and expected worst and best case conditions of the environment on the device under test.

Another aspect of smarter verification is the abstraction of the design under test or the system environment away from simply executing the functionality itself as intended. For instance, when verifying the performance of a design under test, at growing complexity the actual execution of software functions on a processor model in a system may become too slow. Instead these software functions can be replaced by "smarter execution", i.e. pure functional models with performance annotations, that execute much faster, like a natively compiled video coding algorithm as opposed to executing cycle by cycle of its equivalent compiled to the target processor. Similarly, simply executing the system environment - like minutes of video or network traffic – may become too slow depending on the complexity of the environment model. A "smarter execution" will replace brute force execution of the system environment with stochastic representations of the system environment, or pre-recorded traces.



[ Figure 3 – Types of Smarter and Faster Verification ]

One of the key challenges from a prototyping perspective is how the environment or the testbench is executed. In acceleration, the fact that the "smarter" test bench can often not be synthesized is throttling the execution speed of the combined prototype. Various techniques of combining software based simulation and hardware assisted verification are emerging – hybrids of virtual prototype and emulation or "In Circuit Acceleration" as introduced last year combining the best debug capabilities of acceleration with the speed of in-circuit emulation, for example - to address these bottlenecks.

At the end of the day though, we cannot underestimate the subjective effect of actually "seeing" the prototype in action. One of the most complex prototypes I have ever seen in a live demonstration was a phone call made using an LTE phone and LTE network prototype at the Freescale Technology Forum in 2008. Hearing the actual call and seeing actual video transmitted, while not relevant from a perspective of verification completeness, certainly provided confidence that the LTE system that was prototyped did actually work as intended.

In summary - as this article has made clear - the definition of System Prototyping depends on the perspective as well as the scope of what is actually prototyped in combination with the use model what properties the prototype is used for, i.e. to verify hardware, software, their combination or their interaction with the environment. One thing is clear -- with the ever growing complexity of designs and the system environments electronic components have to reside in, technologies for system prototyping will have to continue to evolve, and allow ever faster and smarter execution of verification environments.

#### About The Author:

Frank Schirrmeister, senior director, product marketing, System and Software Realization Group at Cadence Design Systems in San Jose, is responsible for product marketing of the Cadence System Development Suite, accelerating system integration, validation, and bring-up with a set of four connected platforms for concurrent HW/SW design and verification. Prior to Cadence, Frank led the product marketing for systems product portfolio at Synopsys, where he built the strategic plan to expand the firm's product portfolio beyond hardware into embedded software markets, executed and integrated key acquisitions. Other senior roles of Frank's career include VP marketing positions at AXYS Design Automation for processor design and virtual prototyping, Chip Vision Design Automation for low-power system-level design and Imperas for multicore embedded software

design. Frank had originally joined Cadence in 1997 after holding various positions in the area of real time embedded software development, and hardware development of ASICs and IP. At that time he was responsible for the product management of the Felix Initiative – Cadence's first foray into the system-level design and embedded software areas - as well as verification marketing. He holds a Dipl.-Ing. from the Technical University of Berlin.

---

## Accuracy, Now More Than Ever

Bill Neifert, Chief Technology Officer, Carbon Design Systems

If register transfer level (RTL) simulation ran fast enough, you'd use it to accomplish most of your design and verification tasks. Sure, it's an over-simplification of the problem, but think of the things you could accomplish if your RTL simulation ran fast enough.

Need to develop software on your next chip before it's built? Fire up the simulator, hook up a debugger to the virtual JTAG port (let's assume that PLI is fast enough, too) and you're off and running.

Need to try out a few thousand different potential architecture combinations for your next fabric or network on chip? Throw the simulator at them and look at the results a few minutes later.

In reality of course, RTL simulation has nowhere near the horsepower to accomplish these tasks, so there's a long history of products and methodologies from the Electronic Design Automation (EDA) Industry to fill in the gaps. Accelerators, emulators, prototypes (both virtual and hardware based) and even graphics cards have been thrown at the problem in an attempt to get a faster model of the chip before it's fabricated.

Electronic System Level (ESL) approaches have been getting an increasing amount of attention in the last few years as a way to model complex system on chip (SoC) designs. The premise seems promising: gather a bunch of high-level models for the various components in your design, tie them all together into a unified description and distribute it out to all of the various engineering teams to use for development.

Life isn't always as easy as they make it sound in the marketing brochures, however. Pulling together all of these various models typically require a bit more work than it would appear from the PowerPoint slides. In addition, once they're all tied together, the system representation tends to either be too abstract to be used by the users requiring accuracy or too slow to be used by the large pool of software designers requiring speed.

EDA companies are stepping in to streamline the virtual system assembly problem by offering pre-built virtual prototypes, often complete with the software that runs on top of it. The big company

solutions still don't address the issue of meeting designer needs for both accuracy and speed.

To solve this problem, virtual prototypes are starting to become much less virtual because they are being tied to hardware prototypes in the form of an emulator or FPGA board. While this may indeed achieve some of the accuracy goal, it's accomplished by sacrificing a lot of the debuggability, ease of use and affordability that makes virtual prototypes so attractive in the first place. In addition, while EDA companies would love to sell you an emulator for each software user, there aren't many companies out there throwing around that amount of money.

What is needed is not a high-level system design tied to a million dollar emulator, but rather a consistent link to accuracy from within the high-level representation. This can be achieved by combining the model accuracy needed by chip architects with the speed needed by the software developer.

While this approach may sound like a case for approximately timed (AT) virtual models, more often than not, AT models form a compromise unacceptable to users at both ends of the spectrum: They are too inaccurate to make correct design decisions and too slow to be used for software development.

The key to deploying a virtual prototype that's both fast and accurate is to build the prototype using models that are both fast and accurate. An approach successfully deployed in multiple design teams is a model-swap approach. The system executes using fast, abstract models and swaps over to an accurate representation as needed to debug hardware/software interaction or analyze performance. This approach allows the entire prototype to actually remain as a virtual prototype so it can be deployed effectively and inexpensively to all of the users in the design process both inside and outside of the company. It's possible to boot an OS in seconds and still have a completely virtual prototype capable of running with 100% accuracy.

The need for system-level accuracy is not going away. In fact, the need for accuracy at the system level is rising with each new generation of intellectual property (IP). The recent round of cache-coherent bus protocols from ARM, for example, contains nearly 20 different types of read and write transactions. This moves accuracy from the "nice to have" column into "must have." If RTL simulation was fast enough, it could be used to address this problem, but that's far from being the case.

A new generation of problems demands a new generation of solutions. Virtual prototypes can meet this demand for both speed and accuracy without being tied to expensive hardware solutions.

#### About The Author

Bill Neifert, chief technology officer and a Carbon co-founder, has 13 years of electronics engineering experience with more than 18 years in EDA including C-Level Design and Quickturn Systems. Neifert has designed high-performance verification and system integration solutions, and also developed an architecture and coding style for high-performance RTL simulation in C/C++. Neifert has Bachelor of Science and Master of Science degrees in Computer Engineering from Boston University.

---

## System Prototyping and Verification Reuse

Thomas L. Anderson, Vice President, Marketing, Breker Verification Systems

The best place to start an article on system prototyping is to define it. The term is used in many different ways, but most commonly for three specific stages of system-on-chip (SoC) development. The first stage is architectural system prototyping, when SoC architects are defining the basic functionality and performance of the device. The system prototype at this point is abstract, most likely including models of the embedded processors capable of running code and high-level models for major functions of the SoC. The hardware-software boundary is still fluid; defining the division between the two is one of the outcomes of this stage.

Virtual system prototyping is the next stage of refinement. At this stage, decisions have been made on which functionality resides in hardware and which in software. The partitions of SoC functionality are close to the eventual organization of the RTL model, and SoC inputs and outputs are likely identical to the final chip. This model is used by the architects to continue performance studies and design refinements, but also by software engineers to start developing the production code to run on the embedded processors. As with architectural models, virtual system prototypes are purely software so they can be replicated fairly inexpensively across a software team.

Rapid system prototyping moves to a hardware platform, typically an array of FPGAs programmed to replicate the functionality of the SoC. A single FPGA may qualify as an SoC itself these days, so embedded processors and standard interfaces are generally available in a rapid system prototype. Of course, such hardware prototypes have a cost, but the price point may not be very different from a software prototype requiring a commercial simulator and verification IP (VIP) licenses. A project team may buy a few rapid system prototypes for its key programmers in order to perform thorough hardware-software co-verification.

These three types of system prototypes lie along a continuum from early to late stages of an SoC project. The architectural system prototype is the starting point, where the hardware and software are still fuzzy and there's typically not much notion of a testbench. A virtual system prototype has a clear line between the hardware and software, which runs on models of the embedded processors. There is usually a primitive testbench in place to enable the software to perform operations that send or receive data off-chip.

The next stage is testbench simulation with the SoC design evolved into a register-transfer-level (RTL) model. Most projects use constrained-random stimulus generation, most likely under the guidelines of the Universal Verification Methodology (UVM) standard. The focus of simulation is on the testbench and its generated stimulus, with minimal or no code running on the embedded processors. The UVM provides neither embedded code nor any links to the processors. In addition, full SoC simulation tends to be slow. Most SoC teams run little or no code and may replace the processor models with VIP components connected to the processor buses.

Moving from simulation into hardware acceleration can speed verification significantly, although since some of the testbench usually remains in the simulator, the speedup may not be as dramatic as expected. It may be feasible to run some production software on the SoC's embedded processors in acceleration. It is also fairly common for the verification team to hand-write some C tests to ensure that the processors are correctly connected to the rest of the hardware before trying to run production code. However, such tests tend to be simple, perhaps checking a single data-transfer path or verifying that direct memory access (DMA) is operating properly in the major functional units of the SoC.

Contemporary hardware accelerators are also capable of performing in-circuit emulation (ICE), standing in for the SoC and plugging directly into the target system. At this point, any notion of a testbench has disappeared. The emphasis is on running production software, though there may be some handwritten boot-up tests or diagnostics available since it's hard to debug hardware problems discovered when real applications are running. Accelerator/emulator boxes are large and expensive, so it is rare for more than one to be available on an SoC project. Finally, rapid system prototypes also run production software and may also have some handwritten diagnostic tests available.

What's missing in this continuum is effective verification reuse. Production software is not effective at finding hardware bugs. Handwritten tests are costly to develop and inefficient at exercising real use cases. Most of the various models and testbenches developed along the way are discarded. What's needed is a verification technique that fosters reuse through all three stages of system prototyping as well as simulation, acceleration and emulation. Automatic generation of self-verifying C test cases from graph-based scenario models is just such a technique.

Self-verifying test cases run on the embedded processors at every stage of SoC verification and work equally well to verify abstract system models and RTL. They are compact and efficient enough to run in simulation, providing automatic links to testbench components, and move seamlessly to acceleration. The test cases continue to run on the emulator and rapid system prototype, with access to chip inputs and outputs provided through debugger technology. These test cases verify the SoC more thoroughly than handwritten tests, stress-test the design more than production software, and complement UVM testbenches.

This is true verification reuse for system prototyping and beyond.

#### About The Author

Thomas L. Anderson is vice president of Marketing for Breker Verification Systems. His previous positions include Product Management group director of Advanced Verification Systems at Cadence, director of Technical Marketing in the Verification Group at Synopsys, vice president of Applications Engineering at 0-In Design Automation, and vice president of Engineering at Virtual Chips. Anderson has presented more than 100 conference talks and published more than 150 papers and technical articles on such topics as advanced verification, formal analysis, SystemVerilog, and design reuse. He holds a Master of Science degree in Electrical Engineering and Computer Science from MIT and a Bachelor of Science degree in Computer Systems Engineering from the University of Massachusetts at Amherst.

