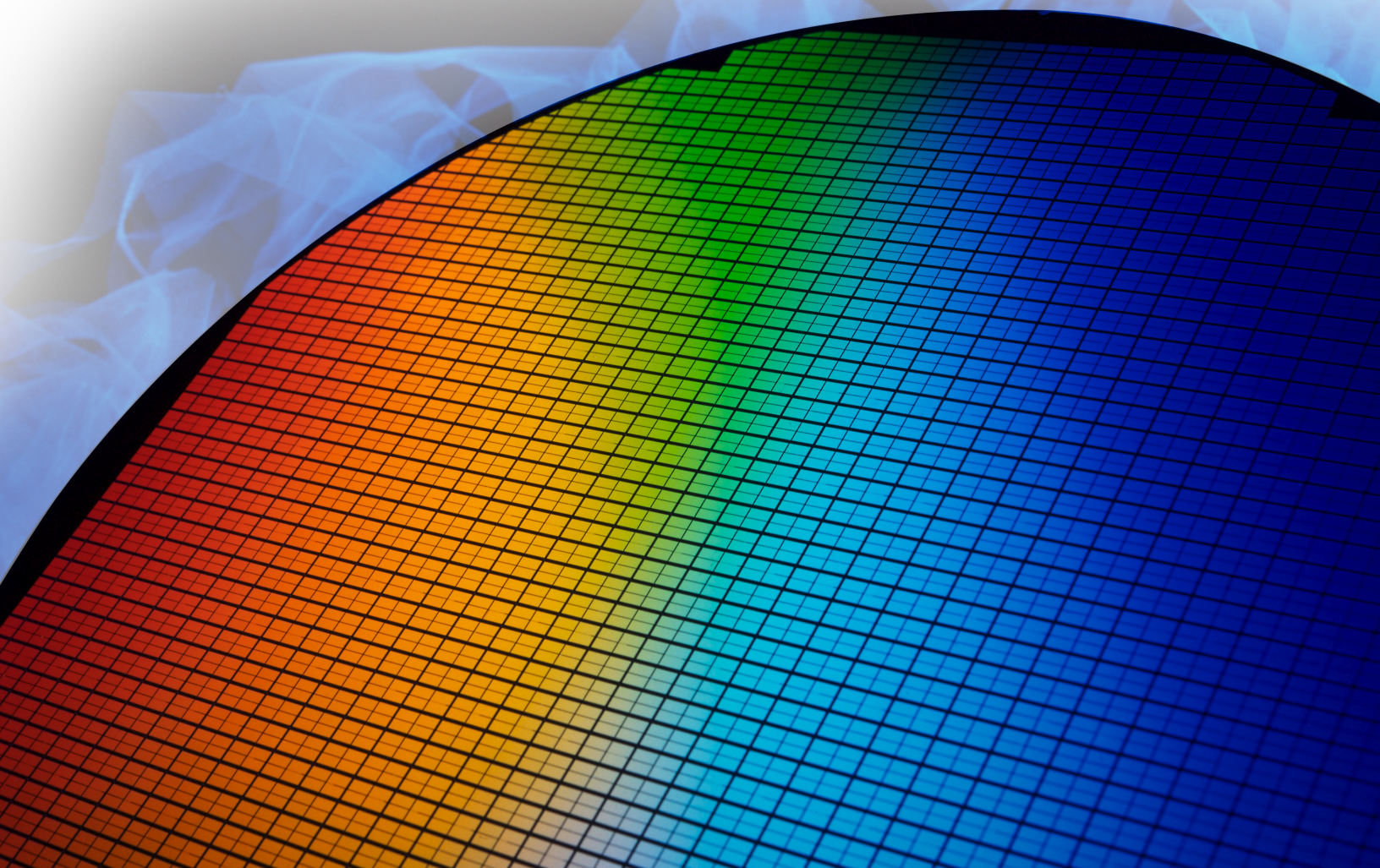
Electronic Design® LIBRARY

A compendium of articles from *Electronic Design*

Challenges and Solutions for Chip Verification



Challenges and Solutions for Chip Verification

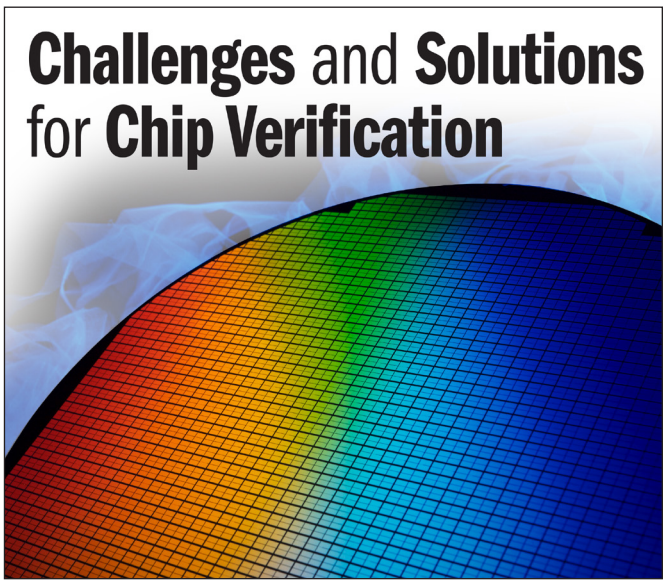


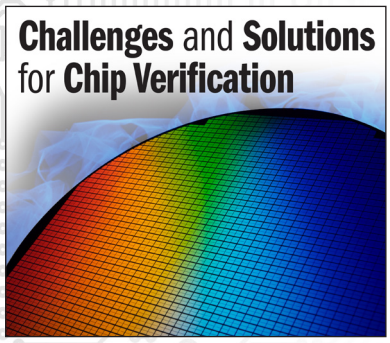
Image: Rainer Plendl | Dreamstime

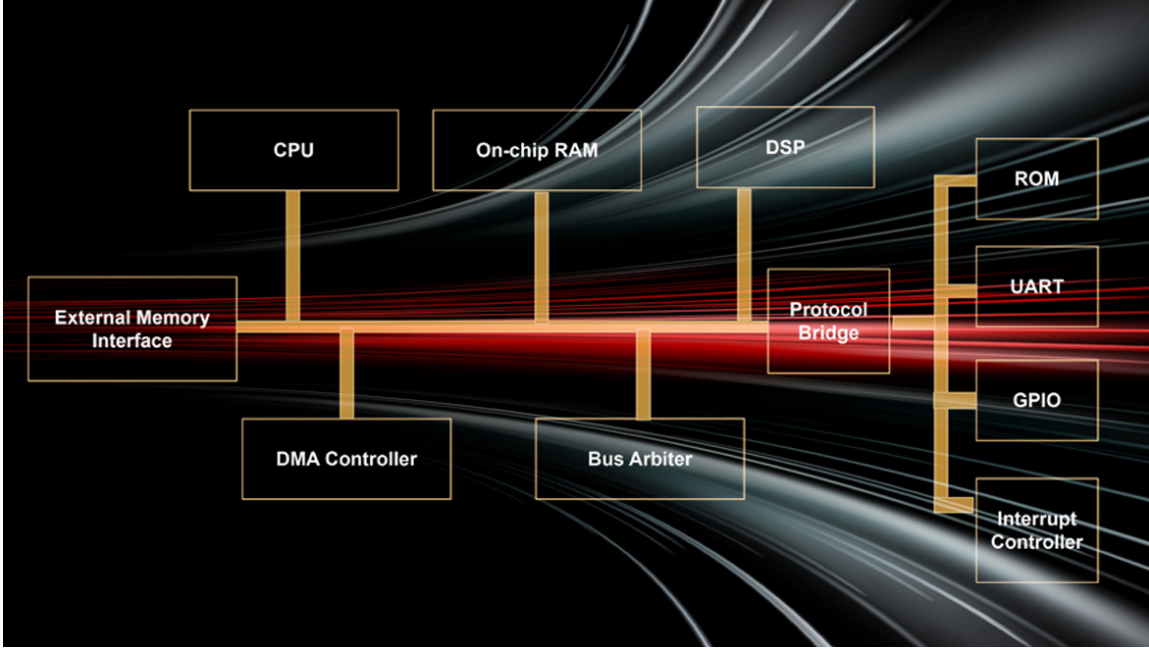
INTRODUCTION

The cost of developing new chips is significant. Verification software and methodologies are important because they help find issues before silicon is generated. Simulation and verification tools are used to check everything from timing closures to thermals. This ebook includes articles from *Electronic Design* that target these aspects of chip verification especially those that deal with power and thermal considerations.

Bill Wong
Editor,
Senior Content
Director, Electronic
Design & MWRF

CONTENTS





All images courtesy of Innergy Systems

CHAPTER 1:

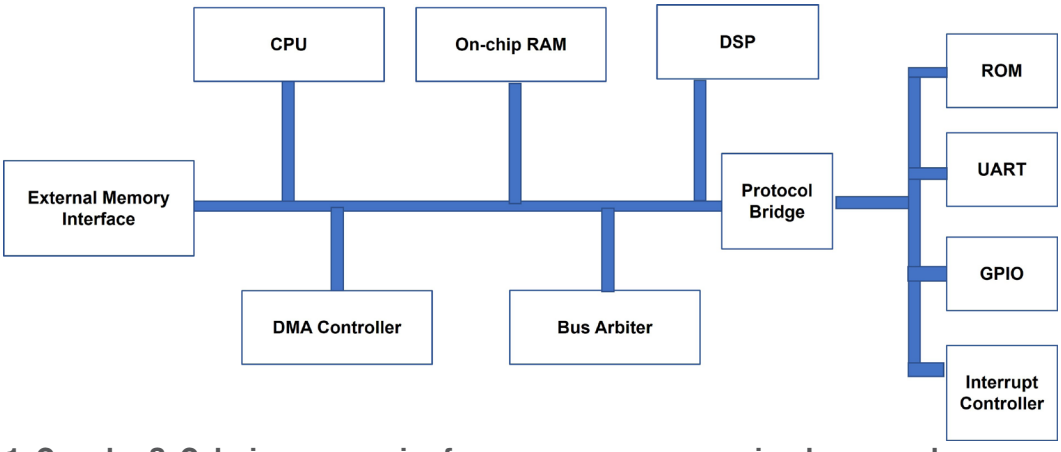
The Need for a New Power-Modeling Approach

NINAD HUILGOL, Founder and CEO, *Innergy Systems*

Power, once considered an all-but afterthought in system design and well behind performance and area, now leads as the first consideration in PPA (power, performance, area).

Low power is now a fundamental performance metric in system-on-chip (SoC) design. Nanometer-scale SoC design complexity—a mix of processor cores, memories, buses, and peripherals with a software stack controlling its operation—runs into tens or even hundreds of millions of logic gates, making power simulation and analysis a daunting task (**Fig. 1**). The result is slow full-chip power simulations.

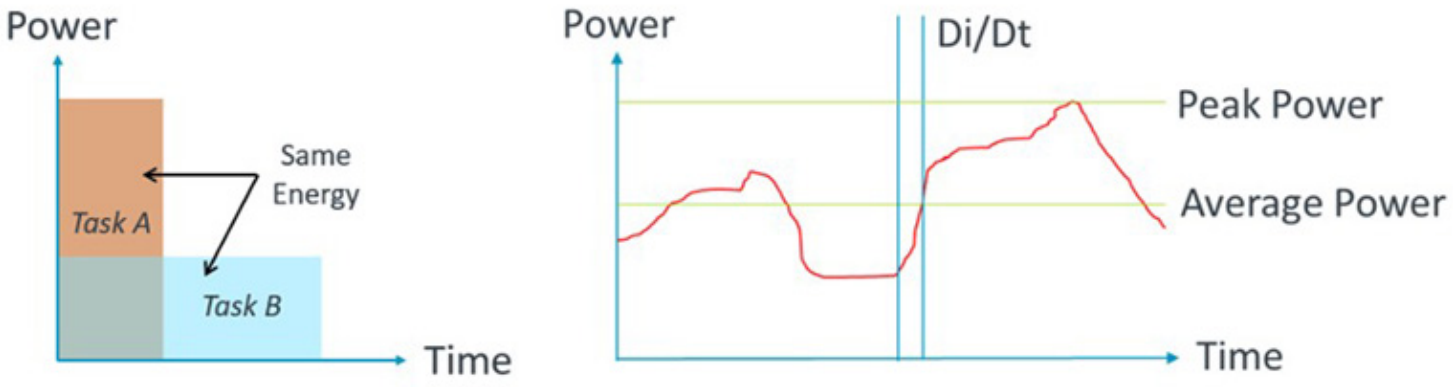
An SoC's power consumption can change in nanoseconds depending on software activity and hardware workloads. That means using design rules, performance of prior generation designs, and methods to estimate power consumption will not give a design's real power footprint. Power hot spots affect reliability, and the choice of chip package can cause catastrophic failure through thermal runaway. A more precise analysis is essential



1. Complex SoC designs are a mix of processor cores, memories, buses, and peripherals with a software stack controlling its operation.

to ensure correct operation and understand a design's real power constraints.

The need to simulate and control dynamic power consumption across a range of stimulus representing real-world conditions and workloads is exploding, along with resources in the SoC deployed to complete a workload task. Tasks and their associated power footprints are managed through hardware or software supervision, control of the number of processors and process threads, and through dynamic changes in voltage, frequency, and bus activity. If a task can be completed using additional clock cycles with fewer hardware resources, then peak power consumption is reduced (**Fig. 2**).



2. The power footprint changes depending on task and level of activity.

Challenges with Today's Power-Analysis Tools

Current power-analysis solutions are focused on the logic gate level and can't provide the throughput to realize comprehensive power visibility at the software level in an SoC. Even hardware emulation can't adequately simulate power consumption for a complete workload task that spans microseconds or milliseconds.

SoC designers have a suite of gate-level and RTL power-analysis tools available for determining power consumption at the block or unit level. Gate-level tools offer sign-off precision while RTL tools provide faster throughput with an accuracy loss that can exceed 10%. For a full-chip SoC simulation, millions or billions of nodes are analyzed for cycle-accurate dynamic power behavior.

RTL tools can't process a full SoC stimulus set without days of runtime (**Table 1**). Due to the lack of speed, engineers typically use only 2% to 3% of the total available stimulus for power analysis (**Fig. 3**). Hardware emulation can only process a portion of a stimulus workload; it's not a general-purpose solution.

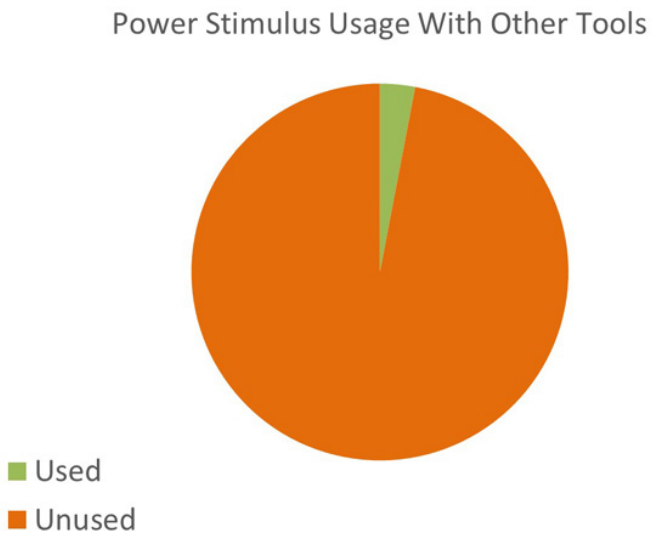
Instead of running the full SoC stimulus available, designers are forced to pick and

1080P Single Frame Power Analysis (GPU)	Full-Chip RTL Simulation	Full-Chip Emulation	Identification of Hot Spots	Resimulation of Blocks	Analysis to Identify Root Cause
17 µS	6 days	5 hours	1 day	1-2 days	1-2 days

Table 1: This table highlights the total time needed for root cause analysis for each approach.

choose where and when to simulate. This is risky since incomplete power verification inevitably invites chip failures, respin, and late software workarounds that can compromise performance and delay the final tapeout.

Another cause of power-analysis delays is identifying root causes of power consumption in the SoC. At the block level, various functions or tasks might be responsible for increased power consumption. At the full chip/SoC level, different design blocks can interact to produce power-density peaks. With the tools currently available, power reports don't estimate power consumption of the functions/tasks or features, which then necessitates lengthy debug and analysis of active signal waveforms.



3. Engineers typically use only 2% to 3% of the total available stimulus for power analysis.

The process of getting power reports, then performing the waveform analysis to identify root causes, can have multiple iterations, increasing the delay in analysis.

What-If

Early "what-if" power analysis is another area that needs attention in every SoC design before making major architecture and design choices. For example, a design architect needs to get early visibility into power consumption by simulating varying transaction density, the effect of increased cache misses, increasing bus throughput for power vs. performance analysis, or changes in voltage and frequency.

"What-if" analysis is an essential step in improving and optimizing a design's power footprint. Consider the following scenario where peak dynamic power reduction at the full-chip level is achieved by stretching the time to complete a task. This is a classical tradeoff that designers perform to understand where power savings can be achieved, improving battery life and reliability, and reducing packaging and cooling costs.

Current power-analysis solutions require numerous manual steps to accomplish many "what-if" power exploration efforts. For example, if the "what-if" power exploration scenario stretches out the time required to complete certain tasks to reduce instantaneous power peaks, and the time spent analyzing power peaks in a GPU (**Table 2**). It further adds time spent modifying the RTL code and re-simulating in a loop that can have many iterations.

Simulation Type	Time Needed
RTL Simulations	9-11 days
Emulation	4-6 days

Table 2: The table illustrates the time typically spent analyzing power hot spots (power peaks) in a GPU at full-chip level.

Given the large number and duration of SoC stimuli, "what-if" analysis is difficult at an SoC's block or unit level. It's almost impossible with the

current power solutions in the full-chip context with a mix of processors, bus activity, and peripherals.

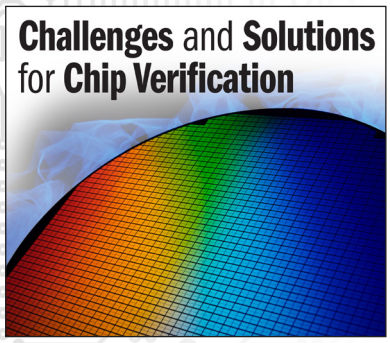
Some SoC designers have taken time to build custom script-ware to do different "what-if" scenarios, explore changes in software activity and hardware resource allocation, and then collect the relevant data and display the desired metrics. These home-grown solutions require effort to build and maintain, are design-specific, and take days of runtime to get results.

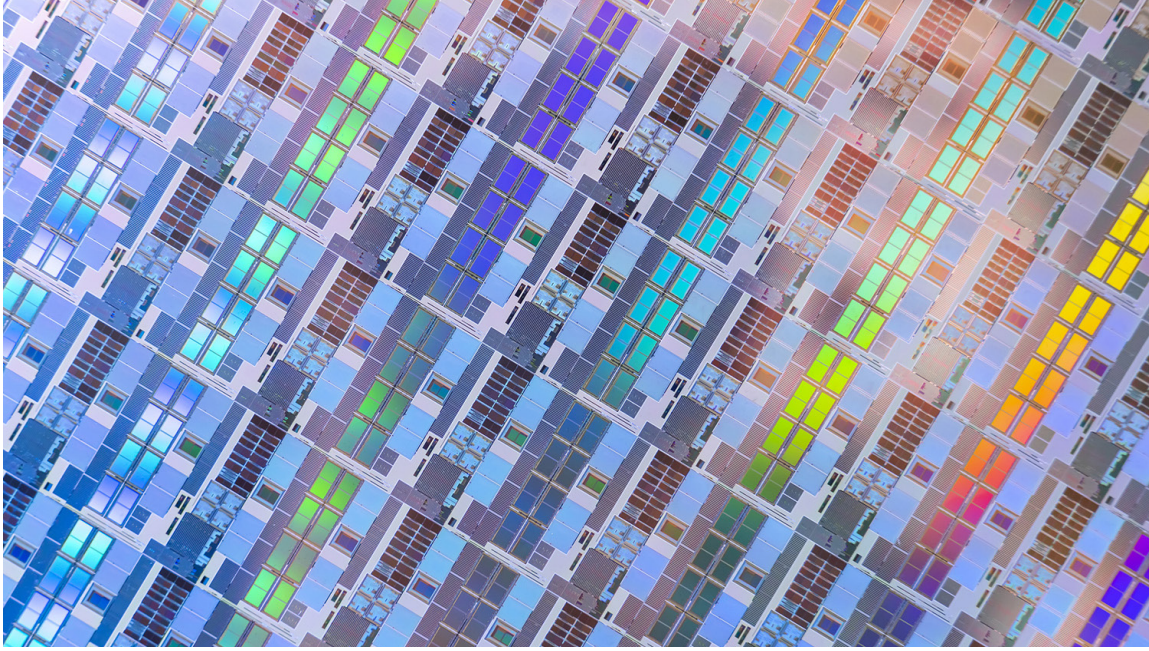
A new power modeling approach is needed to eliminate the SoC power simulation gap and reduce computation time so all full-chip SoC stimulus scenarios can be used to ensure a design's power coverage is 100% and not 2% to 3%. This modeling approach must have precise correlation with results from power sign-off tools.

A tool that shows the root cause of a power issue early in the design flow and highlights those actions contributing to a power hot spot vs. changes in software activity and hardware loads is imperative.

to view this article online, *click here*

BACK TO TABLE OF CONTENTS





Dashark | Dreamstime

CHAPTER 2:

AI Enters the Verification Stage of Chip Design

JAMES MORRA, Senior Staff Editor, *Electronic Design*

The new AI-powered Verisium platform from Cadence helps chipmakers identify problem areas in designs faster.

Cadence rolled out its latest AI-powered electronic design automation (EDA) platform called Verisium, which promises to ease the amount of time and resources that chipmakers put into the verification process.

The Santa Clara, California-based company said MediaTek and Samsung are among the first companies using Verisium to identify bugs in system-on-chip (SoC) designs and diagnose what's causing things to go wrong.

Modern processors are comprised of billions of transistors that must fit into squares of silicon as small as a fingernail. How everything is arranged on the chip and how (and where) it's placed within a system impacts metrics like performance, power efficiency, and even cost. As a result, Cadence has started adding AI to its software tools to automate more aspects of the IC design process.

Verisium is a complement to its Cerebrus Intelligent Chip Explorer platform for AI-enhanced implementation and Optimality Intelligent System Explorer for AI-powered system-level analysis.

A Painstaking Process

The purpose of verification is to identify and resolve chip design defects in a pre-manufactured state. It's the end-stage process of testing the quality of the design and that everything inside works as planned in a product.

The verification process usually starts after you complete the chip design. The hardware is simulated with software code in a hardware description language (HDL) used to test the various building blocks of the SoC. The test bench effectively creates a virtual version of the SoC that can be supplied with signals. Subsequently, you can measure and evaluate the responses from the SoC to figure out whether the SoC or IP inside has any issues.

Cadence said Verisium works with its existing verification engines: Palladium for emula-

tion, Protium for prototyping, Xcelium for simulation, Jasper for formal verification, and its Helium virtual and hybrid studios.

Previously, you would have to run every one of these engines separately for every step in the verification process—what Cadence calls "a single-run, single-engine" approach. Verisium, on the other hand, leverages big data and AI to optimize multiple runs of multiple engines over the full SoC design and verification campaign.

As SoC complexity continues to rise, the verification process tends to take more time and resources than any other silicon engineering task. And so, as Cadence tells it, verification is ripe for improvement using AI.

Verisium also runs on top of Cadence's new "JedAI" platform, which pools vast quantities of data stemming from the chip design process, analyzes it to identify areas of improvement, and even stores it for future use.

Cadence said JedAI is a platform in the sense that its AI-powered offerings—Verisium, Cerebrus, and Optimality—and third-party silicon lifecycle management systems sit on top of it. When it comes to using Verisium, its verification tools feed data stemming from the verification process, ranging from waveforms, coverage, and reports to log files, into the JedAI platform, where it's all stored and evaluated.

Then, JedAI builds machine-learning models and mines other proprietary metrics from the data, sharing what it learns with the company's Verisium to identify potential areas of improvement or root-cause issues.

"As chip design size and complexity has increased exponentially over the past decade, the volume of design and verification data has also increased with it," said Venkat Thanvantri, Cadence's VP of AI R&D. "Previously, we saw that once a chip design project was completed, the valuable data was deleted to make way for the next project. There are valuable learnings in the legacy data, and the Cadence JedAI Platform makes it easy for engineering teams to access these learnings and apply them to future designs."

Starter Apps

Customers can get started with several apps when they use Verisium. Some of them tap into machine learning, both supervised and unsupervised, including reinforcement learning, while others don't.

- **Verisium AutoTriage:** Builds machine-learning models that help automate the repetitive task of sorting through failures to find the worst ones. To do so, it predicts and classifies test failures with common root causes.
- **Verisium SemanticDiff:** Uses algorithms to compare source-code revisions of IP building blocks or the full SoCs. The app classifies these revisions and ranks those that are the most disruptive to the system's behavior to help pinpoint potential bug hotspots.
- **Verisium WaveMiner:** Applies AI engines to analyze waveforms from multiple verification runs and determine which signals, at which times, are most likely to represent the root cause of a test failure.
- **Verisium PinDown:** Integrates with the Cadence JedAI Platform and other industry-standard tools to build machine-learning models of source-code changes, test reports, and log files to predict the source-code check-ins that are most likely to have introduced failures.
- **Verisium Debug:** Natively integrated with the JedAI Platform and other Verisium apps, this app uses AI for the purposes of root-cause analysis, along with support for the

simultaneous and automatic comparison of passing and failing tests. The debug solution spans from IP to SoC and from single-run to multi-run verification.

- Verisium Manager: Brings Cadence's full IP and SoC-level verification management solution with verification planning, job scheduling, and multi-engine coverage onto its JedAI platform. It uses AI technologies to improve how efficiently data centers run verification. This app integrates directly with Cadence's other Verisium apps, opening the door for pushbutton deployment of the complete Verisium platform from a unified browser-based management console.

AI Time Saver

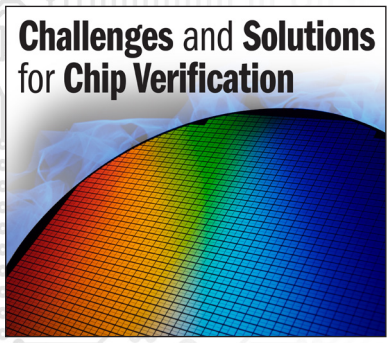
Paul Cunningham, senior vice president and general manager of Cadence's system and verification division, said Verisium would help chip companies make more informed decisions during the design and verification process. But the biggest impact is apparently on the productivity side of things.

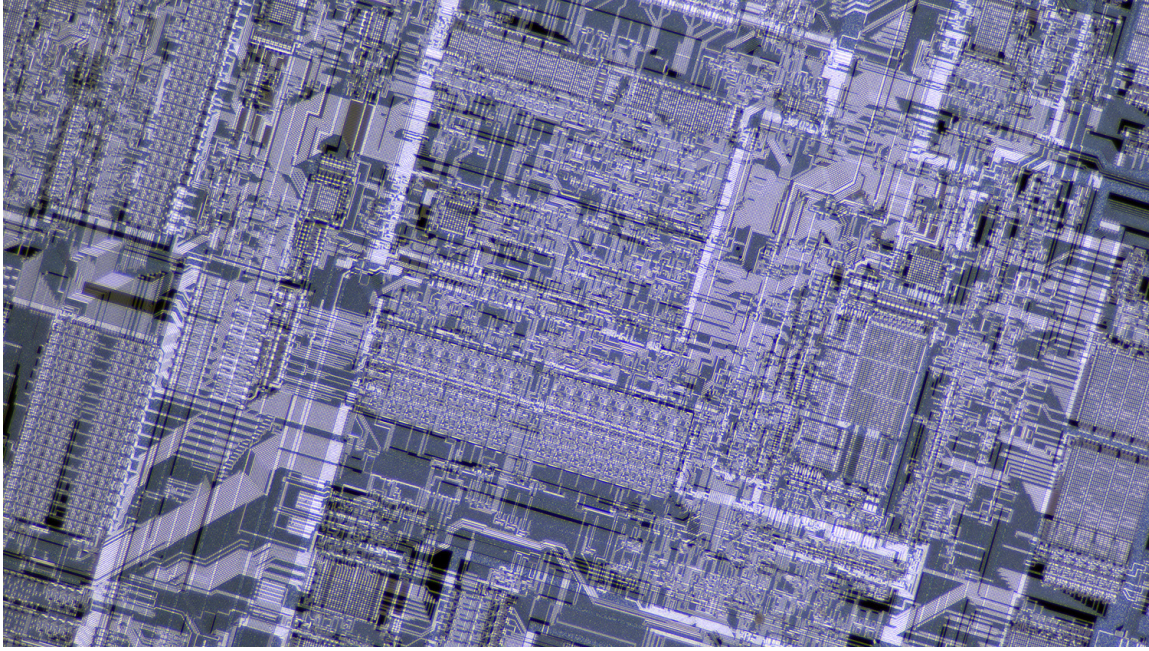
The company said its customers are already using Verisium to triage failing tests more than 3X faster than they could previously, with reductions in the time it takes to determine the root-cause failure by up to 75%.

Given that failure analysis and debug represent 50% (or in some cases more) of the time chip firms devote on verification, Cadence claimed the AI-driven Verisium tool could result in major improvements in productivity.

to view this article online, click here

BACK TO TABLE OF CONTENTS





G00b | Dreamstime

CHAPTER 3:

Pre-Silicon Power Verification for Power-Hungry Applications

DR. JOHANNES STAHL, Senior Director, Product Marketing, Emulation, *Synopsys*

Today's high-performance, power-hungry applications require a new approach to power verification. Emulating power pre-silicon will save weeks of simulation experimentation, delivering a more accurate understanding of power behavior in real-world apps.

5G. Hyperscale data centers. Mobile. Smart everything. These are just a few examples of today's high-performance, power-hungry, or power-constrained applications. Missing your SoC power goals and power bugs on designs like these and others could pack a wallop on overall design success, your bottom line, and time to market. So, how do you optimize for peak and dynamic power and reduce leakage in this era? It's time to shift left via fast power emulation technology and verify the power for designs pre-silicon.

Shifting the Focus in the Power-to-Performance Equation

Over the last 10 years, chip design has been about how fast chips could compute. Each gigahertz of speed represented billion-per-second core operations, a stand in for how fast the arithmetic logic units (ALUs) could process data. The higher the gigahertz, the better, which unfortunately also equates to more compute power consumption.

Historically, chip speed was measured by single- and double-digit percentage improvement rates. For instance, if a chip could run 10% or 20% faster than its predecessor or a competitor, that was considered pretty good.

Today, while clock speed isn't increasing very much, next-generation chips can process many times more data than previous generations through massive parallelism, an entirely different scale of performance improvement. And, as the area of chips becomes smaller, the tradeoffs between compute performance and power are more substantive. That's why in chip design, where achievements in performance are already vast, the conversation is shifting to include a bigger focus on power optimization.

Power-Hungry Designs Drive a New Frontier in Power Verification

Because today's complex designs commonly have billions of gates, power management can be your Achilles heel. The bigger the design, the bigger the risk if you fall short in

managing your power profile well. Some of the applications driving the conversation on power include self-driving cars, networking, mobile, virtual reality, and image recognition—anything that requires processing of massive amounts of data. Here are examples of why some key design categories are putting greater emphasis on power:

- **GPUs:** Running billions of clock cycles per second, power simulation that runs only at key points over time falls short.
- **AI:** Because artificial intelligence is so new, power profiling can be tricky, but it's the holy grail for AI SoC designers to be able to tout both power efficiency and fast compute performance.
- **5G:** Parallel processing at high frequencies—especially in radio-head semiconductors—makes power efficiency critical.
- **Hyperscale data centers:** Billions of gates and complexity in the software workload to boost data throughput to warp-speed levels, with energy efficiency, has put power management front and center.
- **Mobile:** Demand for battery-life longevity in a tiny form factor while performing complex, image-heavy tasks makes power efficiency critical in next-generation designs.

While it's true that these applications have helped create the demand for the exponential leaps in processor speed and performance, they also demand innovation when it comes to power management.

In addition, the climate crisis is making the focus on power an existential one, bringing consumer and regulatory pressure to bear. All of this change requires a lens of precision on power verification—critical to optimizing your power budget and bringing power down.

Why Emulation is the Power Verification Answer for Next-Generation Designs

Designers typically have a phased approach to their work, including architectural exploration, register transfer level (RTL), physical design, and verification. Traditionally, power analysis with realistic software workloads could only be done post-silicon. The outcome of weeks of experimentation to approximate power consumption is heavily influenced by the experience of the engineer, as well as trial and error.

Looking for optimization opportunities in such a way is not only tedious and time-consuming, but accuracy in real-world scenarios is hit and miss. With large, dynamic applications, it's impossible to simulate real-world workloads.

Let's face it, it's easier, quicker, and more budget-friendly to resolve your power issues earlier in the software development lifecycle. If you could detect the power-hungry activities and leakage, optimize dynamic power, and manage for peak power, it would help eliminate risk. Emulation pulls back verification to much earlier in the design process: pre-silicon.

Emulation imitates the actual hardware behaviors and maps the design. It accurately shows the system activity flow so that designers can analyze and benchmark average power for an accurate power profile. Emulation speeds up verification by 1,000-fold over simulation, enabling an exhaustive number of test cycles—millions or even billions of cycles. The result is a much more accurate power profile, with diagnostics showing average power and the components where peaks are high.

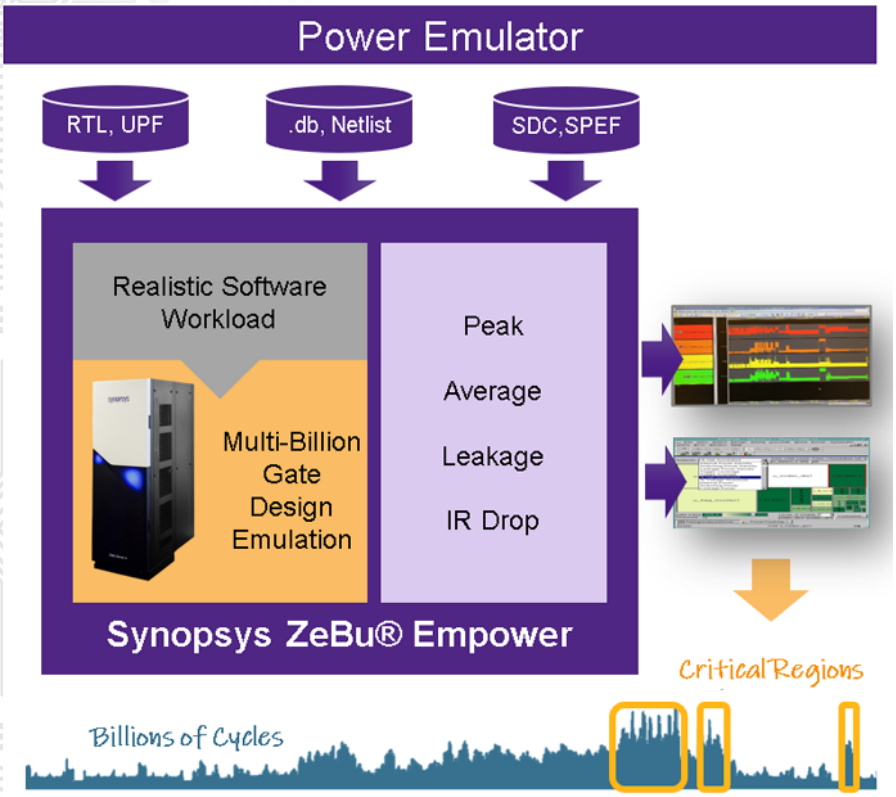
Emulation that Can Verify Power in Hours, Pre-Silicon

Here are a few things to think about when choosing an emulation solution for power

verification. A good emulation system should use real-life workloads pre-silicon and give you power verification within hours. Make sure the design size and emulation cycles can be parallelized, scaling design cycles and tying them to emulation cycles.

A software-driven, low-power emulation solution should reduce overall static and dynamic power consumption. And it will offer maximum compute performance in multiple iterations per day to deliver actionable results. You will be able to identify peak power and improve your design's dynamic and leakage power profiles, easily and early.

Emulation solutions must use fast emulation hardware technologies for quick analysis. They should have adequate capacity and the ability to run in key modes for debugging and pass/fail accuracy **(see figure**).



Shown is an example of power verification using emulation. (Source: Synopsys)

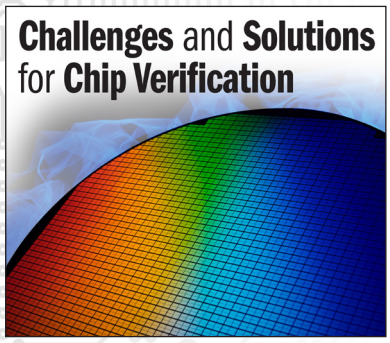
What's Next in the Power-Verification Game?

Gartner projects that new hyperscale data-center infrastructure spending will grow by 6% this year. Also this year, 5G infrastructure revenue will grow by 39%. The mobile-device install base will reach 6.2 billion devices, and with COVID normalizing a new world of remote everything as well as a hybrid work model, that growth can only continue. Finally, AI will generate $2.9 trillion in business value in 2021. Complex, high-performance, power-hungry applications are not only here to stay, but they're also taking us to our future.

With big data only becoming bigger and fueling more applications, power is certainly the next big frontier in SoC designs. The good news is that more accurate, faster power analysis delivering actionable results sooner than ever before is available now. In the future, we can look forward to power analysis in multi-stages of design and development, including utilization of IP blocks with power-consumption models ready to go. With a focus on better methods for power management, the performance of power-hungry applications will only get better.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS



All images courtesy of Axiomise

CHAPTER 4:

11 Myths About Using Formal Verification

DR. ASHISH DARBARI, *Axiomise*

Axiomise's Dr. Ashish Darbari dispels a host of myths to highlight the advantages of formal verification for IC design.

The advantages of formal verification are well-known and accepted in semiconductor development. This wasn't always the case; a few decades ago, formal technology was widely regarded as an exotic technique requiring "magic" to be used successfully on a real project. Over this span, many success stories of truly scary bugs found before signoff have helped to raise awareness of—and confidence in—formal verification.

The ability to mathematically prove exhaustively that a chip design meets a set of assertions is a clear contrast with simulation, which can't come up with proof of bug absence. If proof can't be achieved due to legal design scenarios that violate the assertions, the formal tool presents these as counterexamples and provides information to help designers debug them. Users provide constraints that keep the formal analysis within legal bounds, ensuring that counterexamples are real failure scenarios that could occur in post-silicon chip usage.

This all sounds great, so why isn't everyone running formal verification? It's used successfully every day by thousands of individuals at hundreds of chip and system companies, but some designers and verification engineers are still reluctant. This may be partly due to some persistent myths about formal technology that make it seem too hard or too expensive. This article examines these myths and explains why they should not be cause for concern.

1. You need a PhD to use formal verification.

This myth was arguably true for first-generation formal tools, which were designed for academic purposes. They required learning an obscure mathematical notation to specify assertions and constraints. The tools needed lots of manual guidance, so most users were, in fact, professors and PhD students specializing in formal technology.

Consider the load value axiom for the RISC-V weak memory model. It says that for

threads i, j, and k, if thread i performs a STORE operation followed by thread j performing another STORE, followed by the thread k performing a LOAD, then the value retrieved from the memory by the LOAD will be the latest value updated by the STORE. Formally, a mathematically precise notation can express this as shown in **Figure 1**. However, an average design or verification engineer may not be able to understand these notations, which are friendly to formal method PhDs but not otherwise.

$$
\begin{aligned}
&\forall i, j, k.\\
&\quad \forall t_0, t_1, t_2, \delta_0, \delta_1, \delta_2.\\
&\quad\quad \forall d_i, d_j.\\
&\quad\quad\quad \forall a.\\
&\quad\quad\quad\quad t_0 < t_1 \wedge t_1 < t_2 \wedge t_0 < t_2 \wedge\\
&\quad\quad\quad\quad type(i, t_0) = \texttt{STORE} \wedge data(i, t_0) = d_i \wedge addr(i, t_0) = a \wedge\\
&\quad\quad\quad\quad stable(mem, a, t_0, t_0 + \delta_0) \wedge\\
&\quad\quad\quad\quad t_0 + \delta_0 <= t_1 \wedge\\
&\quad\quad\quad\quad type(j, t_1) = \texttt{STORE} \wedge data(j, t_1) = d_j \wedge addr(j, t_1) = a \wedge\\
&\quad\quad\quad\quad stable(mem, a, t_1, t_1 + \delta_1) \wedge\\
&\quad\quad\quad\quad t_1 + \delta_1 <= t_2 \wedge\\
&\quad\quad\quad\quad type(k, t_2) = \texttt{LOAD} \wedge addr(k, t_2) = a \wedge\\
&\quad\quad\quad\quad\quad \Rightarrow\\
&\quad\quad\quad\quad\quad data(k, t_2 + \delta_2) = d_j
\end{aligned}
$$

1. Example of load value axiom for RISC-V weak memory model.

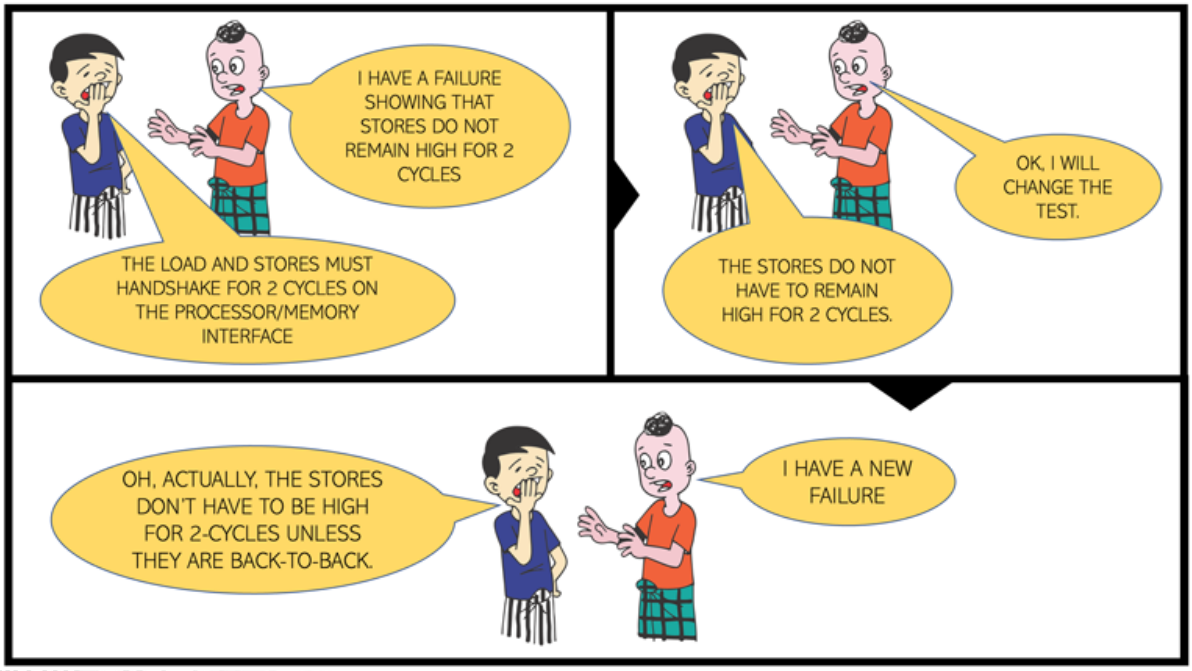
Much has changed in recent years. Assertions and constraints are usually specified with SystemVerilog Assertions (SVA), a subset of the SystemVerilog language that designers and verification engineers already know and use. Formal tools have become smarter and more independent, and less reliant on user expertise. Many now offer visualization and better hints for debugging counterexamples or helping achieve full proofs. No PhD is required.

There's also the whole category of formal applications (apps), which generally don't require users to write any assertions at all. For example, a clock-domain-crossing (CDC) tool can automatically determine where crossings occur in a chip and what assertions must be proven to guarantee correct operation. The user only needs to provide some information on clocks, most of which is already available in the constraint files used by synthesis and layout tools.

2. Formal verification is hard because you need specifications exclusive to formal.

It's not true that specifications are unnecessary for other forms of verification, such as simulation or emulation. The firmware and driver stack in SoC emulation is already providing the right environment to drive stimulus into the chip for testing; checkers rely on requirements to establish what needs to happen when the tests are run. Without specifications, verification engineers can't write directed tests for simulation, the Universal Verification Methodology (UVM), or functional coverage.

Formal methods are perceptibly more sensitive to specifications as the effects of poorly defined requirements are felt more severely. Formal tests, specified as assertions, con-



2. Better specifications are a hidden bargain for formal verification.

straints, and covers, produce unexpected results because formal tools drive all possible combinations of stimulus patterns. This may result in driving spurious stimuli if the constraints captured from requirements aren't accurate.

In many cases, the act of deriving requirements for formal verification from specifications can expose bugs. In fact, a good specification is a hidden bargain for successful formal verification (**Fig. 2**).

3. You can't scale formal techniques to large designs.

This is another myth that was true for earlier generations of formal technology; users were limited to analyzing small design blocks. Today's formal tools have far more capacity, and many are able to run in distributed mode on a server farm or the cloud. Formal techniques and methodologies have scaled up as well.

Designers and verification engineers routinely apply formal verification to large, complex subsystems, including verifying entire multi-threaded 64-bit processors end-to-end with formal. **Figure 3** shows an example of a bug caught by the Axiomise abstraction-based solution in a highly parameterized network on chip (NoC) with over 1 billion gates (338

```
Statistics [for instance "noc_param"]
-------------------------
# Flops:          6153020 (337989788) (656 property flop bits)
# Latches:        0 (0)
# Gates:          6252746 (1017145005)
# Nets:           12423843
# Ports:          7
# RTL Lines:      986
# RTL Instances: 3006
# Embedded Assumptions: 4
# Embedded Assertions:  23
# Embedded Covers:      23
```

A3. This functional bug, caught in a design with more than 1 billion gates, was found by Axiomise using Cadence JasperGold.

million flip-flops).

Formal apps may have even more capacity since they're focused on a single task. CDC analysis, for example, is always run on the full chip to check the entire clock network.

4. Formal proofs take a long time to converge.

This may happen in some cases, especially when the formal test benches haven't been naturally designed to be optimal in performance. However, in most cases, formal properties converge very quickly.

Naturally, the run-time of formal tools depends on design size, design complexity, and the number of assertions and constraints. There are several ways to manage the formal process to keep run-times reasonable. Running incrementally as the design grows and running in a distributed mode both help.

The Axiomise formal-verification training shows users how to apply a range of powerful abstractions to make the analysis mathematically simpler. An exhaustive analysis is, by its very nature, a big task, but users can do a lot to improve efficiency.

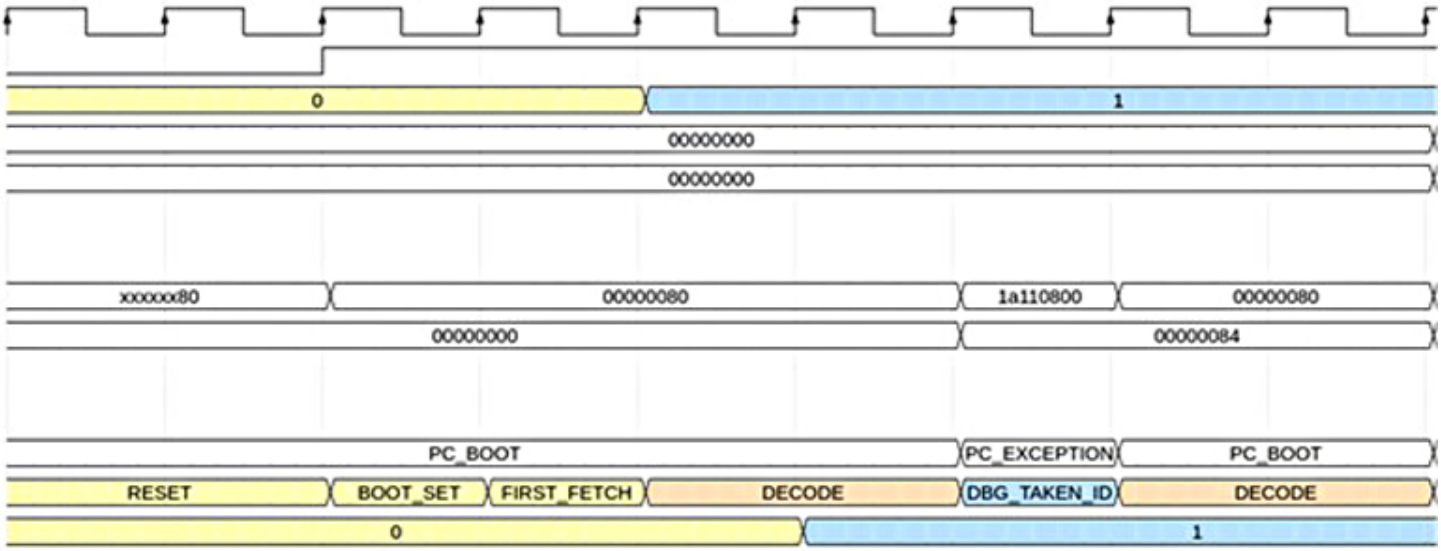
5. Formal techniques are only useful for building proofs.

This myth also derives from academic formal tools where the focus was entirely on achieving full proofs. While full proofs provide the utmost confidence in design correctness, formal verification adds value by finding tough, corner-case bugs such as the example in **Figure 4**.

Name	Groups	Health	Radius	Time
u_isa.axiomise__ISA_SRLI	ISA	10		1m 26s
u_isa.axiomise__ISA_ADDI	ISA	10		2m 40s
u_isa.axiomise__ISA_SRL	ISA	10		2m 40s
u_isa.axiomise__ISA_JALR	ISA	10		2m 46s
u_isa.axiomise__ISA_XORI	ISA	10		2m 46s
u_isa.axiomise__ISA_BEQ	ISA	10		2m 53s
u_isa.axiomise__ISA_BNE	ISA	10		2m 59s
u_isa.axiomise__ISA_SLTSI_SET_TO_1	ISA	10		3m 0s
u_isa.axiomise__ISA_BGEU	ISA	10		3m 6s
u_isa.axiomise__ISA_SLTSI_SET_TO_0	ISA	10		3m 6s
u_isa.axiomise__ISA_XOR	ISA	10		3m 7s
u_isa.axiomise__ISA_SLTUI_SET_TO_1	ISA	10		3m 13s
u_isa.axiomise__ISA_OR	ISA	10		3m 13s
u_isa.axiomise__ISA_ORI	ISA	10		3m 19s
u_isa.axiomise__ISA_SLTUI_SET_TO_0	ISA	10		3m 19s
u_isa.axiomise__ISA_ANDI	ISA	10		3m 25s
u_isa.axiomise__ISA_SLLI	ISA	10		3m 25s
u_isa.axiomise__ISA_SLTS_SET_TO_1	ISA	10		3m 27s
u_isa.axiomise__ISA_SLTU_SET_TO_1	ISA	10		3m 27s
u_isa.axiomise__ISA_BLTS	ISA	10		3m 32s
u_isa.axiomise__ISA_SRAI	ISA	10		3m 32s
u_isa.axiomise__ISA_SLTS_SET_TO_0	ISA	10		3m 33s
u_isa.axiomise__ISA_SLTU_SET_TO_0	ISA	10		3m 33s
u_isa.axiomise__ISA_JAL	ISA	10		4m 26s
u_isa.axiomise__ISA_LUI	ISA	10		4m 32s
u_isa.axiomise__ISA_BGES	ISA	10		4m 45s
u_isa.axiomise__ISA_SRA	ISA	10		4m 45s
u_isa.axiomise__ISA_BLTU	ISA	10		4m 52s
u_isa.axiomise__ISA_ADD	ISA	10		4m 58s
u_isa.axiomise__ISA_AND	ISA	10		4m 58s
u_isa.axiomise__ISA_SUB	ISA	0		5m 5s
u_isa.axiomise__ISA_SLL	ISA	10		5m 31s
u_isa.axiomise__ISA_JAL_ret_address	ISA	10		24m 13s

~50% exhaustively proven in less than 30 min!

4. End-to-end RISC-V formal verification: 50% complete in under 30 minutes using the Axiomise formalISA app and, in this case, QuestaPropCheck from Siemens.



5. BEQ instruction failure due to a bug in ibex RISC-V core, triggered by incoming debug requests only when the FSM controller is in the DECODE state.

The waveform in **Figure 5** shows the bug caught in ibex RISC-V core using the Axiomise formal-verification solution. This bug will only ever appear in the design if the debug request arrived in the same clock cycle when the controller FSM was in DECODE state. The bug will not show up in any other state. The precise timing of the arrival of debug will make this kind of bug very hard to catch with dynamic simulation, where controllability of stimulus and exhaustive coverage would be a major challenge.

6. If you have run the simulation with 100% coverage, you don't need formal techniques.

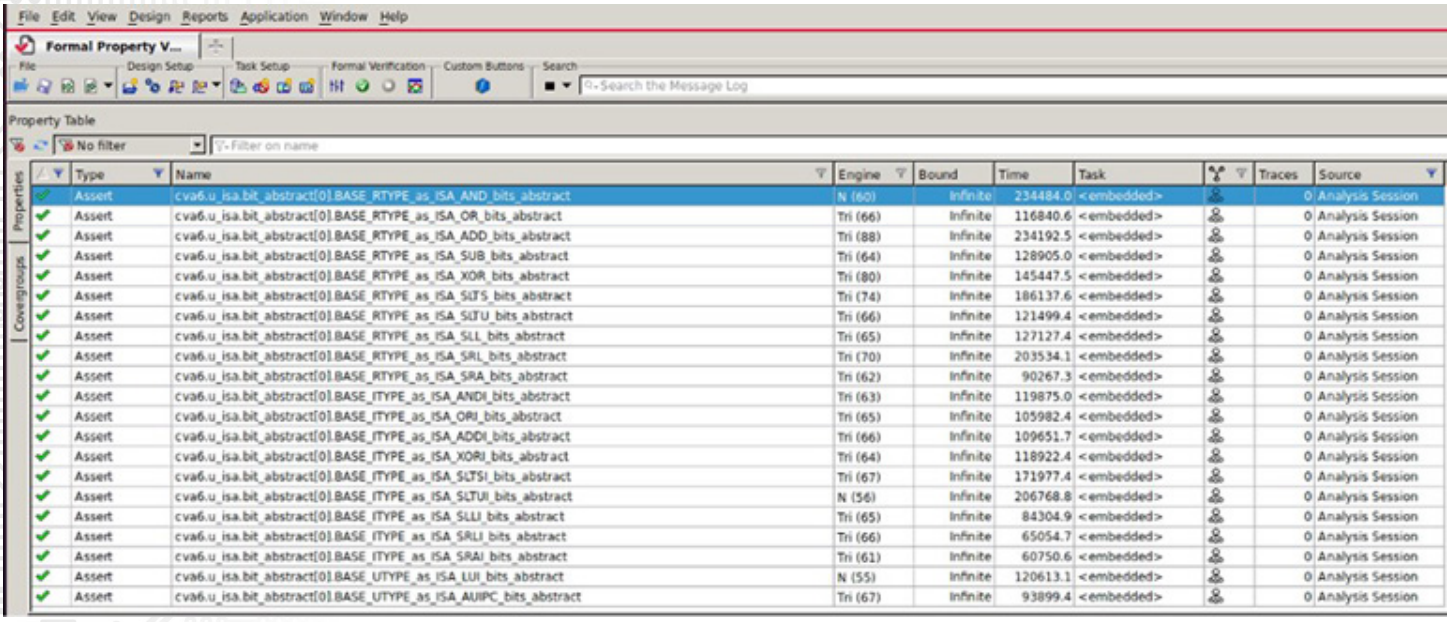
As noted earlier, formal verification is great at finding corner-case bugs missed by simulation or emulation. In addition, this myth rather overstates the value of coverage metrics. They are extremely valuable in identifying parts of the design not yet exercised, and there's no chance of finding all bugs in such a case.

However, also as noted earlier, simulation can't establish exhaustive mathematical proofs. Even 100% functional coverage doesn't guarantee no bug escapes—it just confirms the exercise of the parts of the design covered by the selected metrics. The formal analysis will consider all possible behavior and is highly likely to find additional bugs.

7. Formal techniques are only useful for finding corner-case bugs.

Many formal users swear by formal for bug hunting, sometimes to such an extent that their management believes that formal is only good for bug hunting. One of the greatest benefits of formal is in establishing that bugs do not exist in the design with respect to the formally proven requirements.

Consider RISC-V, for example. A lot of the processors previously verified with simulation end up having bug escapes that are then caught by formal. Formal can prove beyond doubt that once bug fixes have been applied, there are no bugs remaining as formal properties prove on all reachable states of the design (**Fig. 6**).



Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_AND_bits_abstract	N (60)	Infinite	234484.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_OR_bits_abstract	Tri (66)	Infinite	116840.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_ADD_bits_abstract	Tri (88)	Infinite	234192.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SUB_bits_abstract	Tri (64)	Infinite	128905.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_XOR_bits_abstract	Tri (80)	Infinite	145447.5	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTS_bits_abstract	Tri (74)	Infinite	186137.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLTU_bits_abstract	Tri (66)	Infinite	121499.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SLL_bits_abstract	Tri (65)	Infinite	127127.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRL_bits_abstract	Tri (70)	Infinite	203534.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_RTYPE_as_ISA_SRA_bits_abstract	Tri (62)	Infinite	90267.3	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ANDI_bits_abstract	Tri (63)	Infinite	119875.0	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ORI_bits_abstract	Tri (65)	Infinite	105982.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_ADDI_bits_abstract	Tri (66)	Infinite	109651.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_XORI_bits_abstract	Tri (64)	Infinite	118922.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTSI_bits_abstract	Tri (67)	Infinite	171977.4	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLTUI_bits_abstract	N (56)	Infinite	206768.8	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SLLI_bits_abstract	Tri (65)	Infinite	84304.9	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRLI_bits_abstract	Tri (66)	Infinite	65054.7	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_ITYPE_as_ISA_SRAI_bits_abstract	Tri (61)	Infinite	60750.6	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_LUI_bits_abstract	N (55)	Infinite	120613.1	<embedded>	0	Analysis Session
Assert	cva6.u_isa.bit_abstract[0].BASE_UTYPE_as_ISA_AUIPC_bits_abstract	Tri (67)	Infinite	93899.4	<embedded>	0	Analysis Session

6. The dashboard shows how the Axiomise formalISA app, working with JasperGold in this case, can be used to find bugs and build architectural proofs of correctness for end-to-end verification of 64-bit RISC-V processors.

It's certainly true that there's no better demonstration of formal power than finding a deep, scary bug that would have required a chip turn. A verification engineer saying "we never would have found that in the simulation" quickly makes believers of formal.

But formal verification can find all sorts of bugs, including those typically uncovered in simulation, more rapidly. For this reason, today's chip projects often contain multiple blocks, some of them quite large, verified formally without any block-level simulation.

8. Once you have applied formal techniques, you don't need to simulate.

Typically, every formal-verification environment uses constraints to describe the interfaces. These constraints need to be validated in simulation to check if they were modeled and interpreted correctly for formal verification.

Also, formal is usually applied earlier in the flows to get the maximum value for shift-left of verification. By the time the design has matured with more blocks being coded, it's possible that some of the interface constraints may no longer be valid, so they must be re-validated in simulation.

Furthermore, simulation and formal are valuable in finding bugs related to hardware-software interactions that occur only in simulation or emulation when the software is running on embedded or host processors. Similarly, bugs along the analog-digital interface may only be found when running mixed-signal simulations.

9. Formal techniques don't offer any coverage metrics, so it's hard to know if you have done enough.

This is manifestly untrue since proofs provide one form of coverage metric. Knowing that 100% of the assertions in a design can never be violated is clearly a strong statement.

However, all modern tools now produce a code-coverage view in relation to proven asserts in formal showcasing (**Fig. 7**). It shows which lines of design code were activated

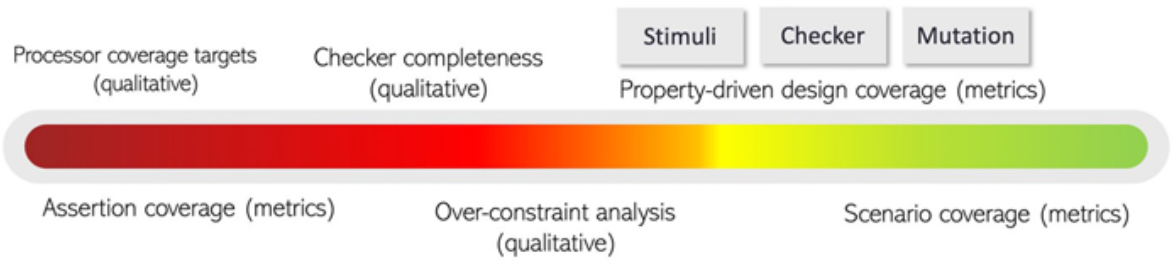
Instance	Ex.	Checker Coverage
cv32e40p_core (cv32e40p_core)		15288/18692 (81.79%)
sleep_unit_i (cv32e40p_sleep_unit)		36/47 (76.60%)
if_stage_i (cv32e40p_if_stage)		2145/2552 (84.05%)
id_stage_i (cv32e40p_id_stage:(PULP_XPULP=0,WAPUTYPE=1))		4760/5599 (85.02%)
ex_stage_i (cv32e40p_ex_stage)		4672/5210 (89.67%)
load_store_unit_i (cv32e40p_load_store_unit)		1003/1075 (93.30%)
cs_registers_i (cv32e40p_cs_registers)		1524/2472 (61.65%)

7. Checker coverage shown by the JasperGold Coverage app for the 32-bit cv32e40p processor verified with Axiomise formalISA app for RISC-V.

and run during the formal proof.

Formal tools were used previously to evaluate code coverage in the absence of any checkers in formal. They could still provide insights into unreachable and dead code, possibly as a result of conflicting design code or configurations. Formal tools also are used extensively to prove that unreachable code-coverage holes in UVM environments may be always unreachable or may find a coverage gap in UVM.

The six-dimensional coverage flow developed by Axiomise describes how coverage for formal can be computed both qualitatively and quantitatively (**Fig. 8**).



8. Six dimensions of formal coverage.

10. Simulation and formal verification can't be combined.

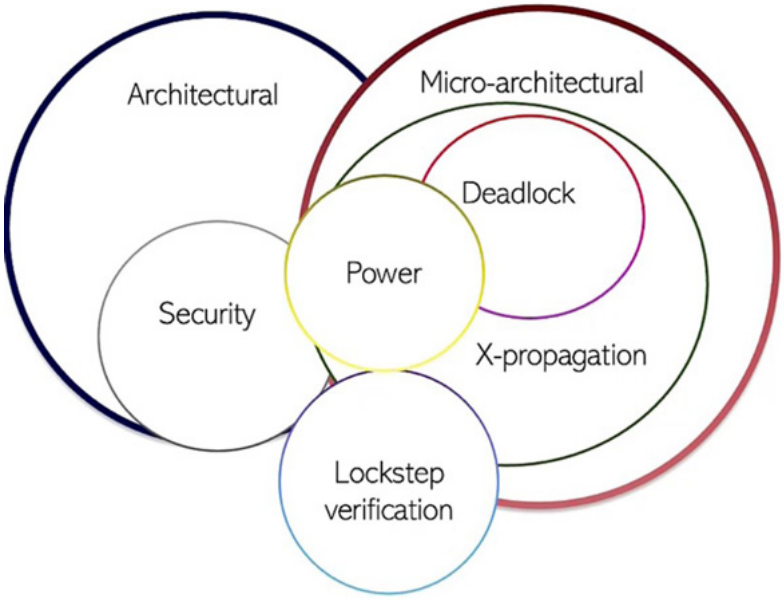
As discussed earlier, the two verification approaches are complementary. Each can find certain types of bugs that the other likely will not. No chip project runs one without the other. Think of this as assuming interface assumptions to guarantee bug absence for blocks in formal verification and then validating assumptions in simulation to close the complete loop.

In addition, using formal to establish coverage gaps in simulation is a great example of combining the two technologies. Many project-management tools that track coverage results over time gather metrics from both simulation and formal verification to present a unified view of verification progress. This helps to convince the boss that the team is meeting the requirements of metrics-driven verification.

11. Formal techniques are only useful for functional verification.

The general concepts of assertions, constraints, exhaustive mathematical analysis, proofs, and counterexamples show up in areas of chip development beyond checking functional correctness.

Today, formal tools are extensively deployed for verifying architectural requirements,



9. Formal verification's pervasive usage.

CDCs, connectivity, power, deadlock, micro-architectural functional requirements, safety, security, and X-propagation (**Fig. 9**).

A recent example presented at DAC 2021 showed how formal verification could be used to find security vulnerabilities (confidentiality, integrity, and availability) in RISC-V cores and rank them with a vulnerability score. The greatest challenge with security is dealing with unknown attack scenarios. This is where formal really shines as it introduces all sorts of input stimuli in an attempt to be exhaustive, finding scenarios that designers would normally never consider.

The act of deploying formal forces designers and architects to think of exploiting vulnerabilities during early stages of architecture development, avoiding any ugly surprises downstream.

Formal techniques are central to the successful design, verification, and implementation of today's chips. With the 11 myths dispelled, no one should hesitate for a second about embracing formal verification and other formal technology.

to view this article online, ☞ *click here*

☞ BACK TO TABLE OF CONTENTS