

Feature Story

## Formal property checking -- what the users say

By Richard Goering

02/19/08

Formal property checking and "bug hunting" tools promise to find tough corner-case bugs and provide exhaustive proofs. But what's the reality? In the first part of this two-part report, engineers at IBM, Infineon, D. E. Shaw Research, MIPS Technologies, and Sun Microsystems discuss the advantages and limitations of existing formal tools.

Originally referred to as "model checking" tools, formal property checkers mathematically prove whether the behavior of a design conforms to the specification, expressed as a set of property descriptions. Two types of formal tools are described in this report – static property checking tools that do not use simulation, and dynamic or "hybrid" tools that run with simulation.

To use most formal property checking tools, engineers write properties that are expressed as assertions. Tools increasingly support standard languages, most notably SystemVerilog assertions (SVA), which can also be used for simulation. Some tools offer automated checks, such as clock-domain crossing checks (CDC), which reduce or eliminate the need to write properties.

Static property checking tools provide exhaustive proofs. They typically prove either that a property holds or that it fails, or they identify an incomplete proof, which may require user intervention to modify the property or re-partition the logic.

Dynamic formal tools generally don't provide exhaustive proofs, and are used primarily for bug hunting. To use these tools, engineers typically first run simulation to bring the design to a particular state, and then use the formal tool to dive deep into the target state space. Users say that dynamic tools can find corner case bugs that simulation alone cannot, and can find coverage points that are unreachable by simulation.

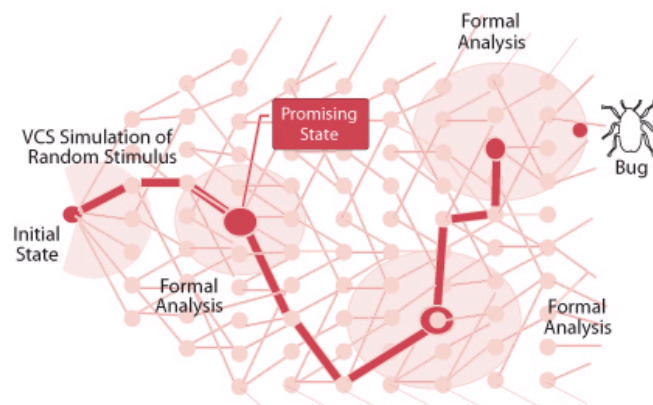


Figure 1 – Dynamic or "hybrid" formal tools such as Synopsys' Magellan work with simulation to search deep in the state space for corner-case bugs. (Source: Synopsys)

Gary Smith, chief analyst at [Gary Smith EDA](#), divides the formal tool market into "formal analysis" and "formal verification." Formal analysis tools use formal methods but don't provide complete proofs. They include "assertion-based verification" tools that ride on top of simulators. Formal verification tools provide complete proofs. "They are both strong tool types with a good following. All of the power users use them, and a good chunk of the upper mainstream," Smith said.

### Pros and cons

Formal tools can verify a variety of specified or allowable conditions. They can verify a protocol, find deadlock or livelock conditions, make sure that an arbiter never misses, ensure that priorities aren't violated, or make sure that a FIFO is never driven when full. Some can check "end to end" specifications, such as ensuring that packet conversion hardware produces the right code at the outputs.

Why formal? Harry Foster, chief verification technologist at [Mentor Graphics](#), offers a striking example. Take a 32-bit comparator, he says, and try to exhaustively prove it has no bugs. Using simulation, this would take  $2^{64}$  test vectors. If you run one test vector every microsecond, which

would be a fast simulation, it will take 600,000 years to verify it. "I can formally prove that [comparator] in less than one second with any tool on the market," Foster said.

Those who use formal property checking tools on a regular basis appreciate their power. "In my mind, the debate over the value proposition of formal is over," said Viresh Paruthi, technical lead for formal verification at [IBM Systems Group](#). "It's clear from everybody's experience that this is a technology that should be leveraged to improve design and verification productivity."

"Even if you do leading-edge verification with constrained-random testbench automation and coverage-driven verification, you still have verification holes," said Claudia Blank, senior staff engineer for formal verification at [Infineon Technologies](#). "There are always corner cases that aren't covered even if you have a very good coverage model. With property checking, you can really go into every corner of the design."

But there are challenges. Chief among these are the capacity limits of static property checking tools, and the possibility of a state space explosion if a block is too large or properties are too complex. And when proofs fail to complete, expertise is often needed to make them work.

Formal verification, said verification consultant [Brian Bailey](#), "is still in the realm of the experts. For all of the promises, it's never become an easy thing to understand and use."

Bailey said that the ease of use of formal tools has "improved a lot" in recent years, and that standard assertion languages are making it easier to write properties. But when it comes to capacity, he said, the tools aren't keeping up with increasing design sizes.

Where formal tools are most successful, Bailey said, is with applications that run the risk of catastrophic failure. "If I'm a company in China making low-cost toys, I wouldn't dream of using formal," he said.

"It's very easy to fail with formal tools," said Richard Ho, who verifies high-end ASICs destined for supercomputers at computational biochemistry laboratory [D.E. Shaw Research](#). "It requires real dedication to make them work right now. I believe it does require dedicated specialists."

### Users cite concerns

Some of the reasons for user hesitancy around formal property checking tools became apparent in a [2007 survey of 818 verification engineers](#) conducted by John Cooley. The survey asked engineers whether their project used "formal bug hunters." 74.5 percent of respondents said no. Of those who said yes, the most frequently used were Synopsys Magellan, Mentor Graphics O-In, Cadence Incisive Formal Verifier (IFV), Jasper Design Automation JasperGold, and IBM RuleBase.

Some respondents had positive things to say about formal tools, noting that they can be extremely useful for finding tough corner-case bugs. But there were also complaints that the tools are difficult to use, time-consuming to set up, useful only for small portions of designs, produce false positives, don't work for datapaths, and require expertise. Some of the commentators said that formal tools are a "waste of time."

The formal verification users represented in this report clearly do not agree that formal property checking is a "waste of time." All find great value in formal tools. But they provide a realistic view of tools that have both benefits and challenges, and require experience and expertise to use to their fullest advantage.

Formal property checking tools mentioned by users in the following sections include:

- Mentor Graphics [O-In](#) – a formal verification suite including static and dynamic capabilities, automated CDC, and an assertion library.
- Jasper Design Automation [JasperGold](#) – a static property checker ranging from assertion-based verification to end-to-end, micro-architectural level properties.
- Synopsys [Magellan](#) – a hybrid formal tool that runs with simulation to detect bugs; can also run in static mode.
- Cadence Design Systems [IFV](#) – a static property checker that supports the same assertions as Incisive simulation, coverage, acceleration, and emulation.
- OneSpin Solutions [360MV](#) – a static property checker that leverages both global transaction-level properties and local assertions.
- IBM [RuleBase PE](#) – a static property checking tool with some dynamic capabilities. It is used internally and sold outside of IBM.
- IBM [SixthSense](#) – a static property checking tool used inside IBM.
- Real Intent [Envision](#) – a static property checking tool suite with both automated and user-provided properties.

### Sun: A range of formal tools

[Sun Microsystems](#) is serious about formal verification. The company uses formal tools "in every category," according to Shrenik Mehta, senior director of front-end technology at Sun. This includes not only static and dynamic formal property checking tools, but also equivalence checking, symbolic simulation, and clock domain checking. The tools come from commercial vendors and academia, but Mehta declined to identify suppliers.

Catherine Ahlshlager, hardware manager for Sun Microsystems' formal technology group, heads up a team of over 10 people that supports a variety of formal tools. While formal property checking tools are much easier to set up than they were five or six years ago, there's still a need for formal

verification specialists. Ahlshlager said that, as a proof fails, she noted, a lot of debugging can be involved. When a proof is inconclusive, "then you have to interact with the tool. That's a little bit involved, and that's the competency of our group."

Ahlshlager said that Sun uses formal tools for bug hunting, RTL code checking, and module-level property verification. Two other applications are architectural verification and post-silicon debug. Sun uses both static and dynamic formal property checking tools, and there are advantages and tradeoffs for both, Ahlshlager said.

Dynamic or hybrid tools, she said, can handle larger blocks than purely static tools. For large blocks, she said, Sun engineers use the dynamic approach as a first choice. "When we do a static approach, if we can narrow it down to as small a block as possible, we have a better chance of getting convergence," she said. But the dynamic approach is mostly for bug hunting, while the static approach can yield conclusive proofs, she noted.

Engineers used to simulation might have some difficulties debugging with static tools. "The waveforms that static tools tend to give you are for the shortest paths," Ahlshlager said. "Interpreting the waveform is counter-intuitive if you're used to debugging in the normal simulation environment. It just takes a while to get used to."

Sun uses SVA to write assertions. Ahlshlager noted that any property engineers write for formal tools can run in simulation, but the reverse isn't necessarily true. Properties written for simulation may involve too much latency or too much logic for formal tools to handle. Further, to prove a property in a formal tool, it's necessary to apply input constraints, Ahlshlager said. "You have to make sure the stimulus coming in to drive the proofs is real."

While the capacity of commercial formal tools is improving every year, there can still be challenges with deep counters, FIFOs, or embedded memories, Mehta noted. He said Sun engineers look to "find the domains where we can depend on formal to provide a complete solution."

With the UltraSparc T2, Ahlshlager said, Sun used formal verification on about 80 percent of the blocks, "but not the entire functionality of the blocks. We picked and chose features that were important to us." For example, she said, formal property checking was used to ensure that threads are always productive, and that threads are not redirected during a pipeline flush. Property checking was applied in the processor's core, crossbar, and memory subsystem (see Figure 2).

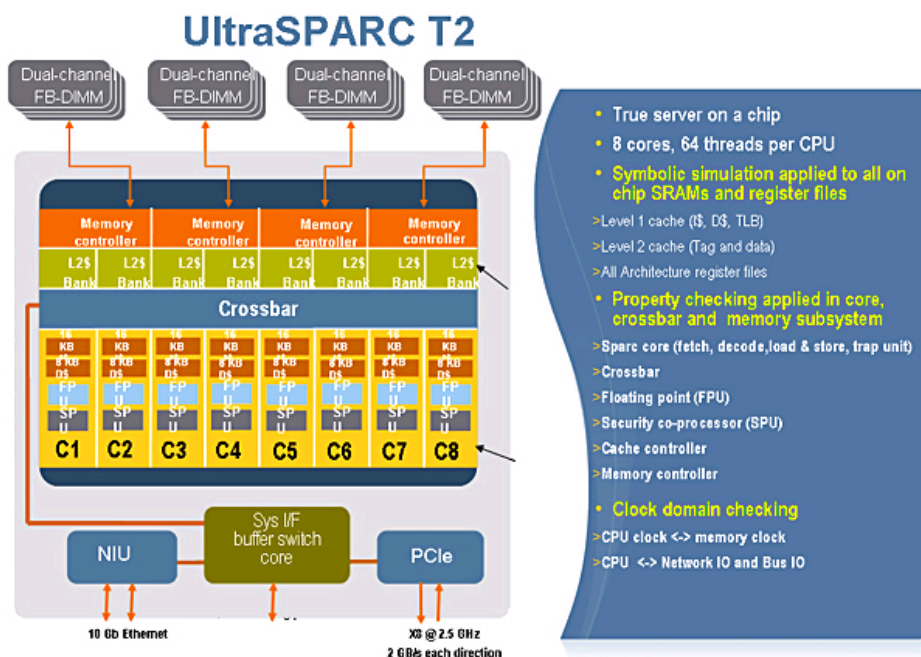


Figure 2: Formal verification was used extensively for the 8-core Sun UltraSparc 2. (Source: Sun Microsystems)

What would Sun most like to see improved with formal tools? "It's all about scaling, capacity, and speed," Mehta said. "Capacity needs to be addressed on an ongoing basis." A second concern, he said, is ease of use, so that new design and verification engineers can ramp up quickly with formal tools. Sun also wants to see unified coverage metrics for formal verification and simulation. Mehta is pursuing that goal as chairman of Accellera, which is sponsoring the Unified Coverage Interoperability Standard (UCIS) working group.

#### D.E. Shaw: A need for specialists

D.E. Shaw Research's Ho has a long background in formal tools. He was a co-founder of O-In before its acquisition by Mentor Graphics, and he served as O-In's chief architect. Today, Ho uses O-In tools from Mentor and JasperGold from Jasper Design Automation in his verification work with high-end ASICs.

"We have a couple of specialists, myself being one of them, and we're trying to get verification engineers as well as designers to use some aspects of the tools," Ho said. "We found it's critical to have at least one specialist trained in formal tools. Otherwise they end up sitting on servers and

not being deployed as well as they could be."

The team at D.E. Shaw, Ho said, is trying to get the most effective use from formal tools in a simulation-based environment. The key, he said, is using formal tools only where they're most effective. "We've given up on the idea of formally verifying everything," he said. "We go only after the sweet spots and leave everything else to simulation. It's more cost-effective for our engineering and CPU time."

Ho said his company uses formal tools for bug hunting, RTL code verification, and end-to-end property checks. He noted that engineers use formal tools for datapaths as well as control logic. The best approach, he said, is to identify up front where simulation might have difficulty, and to apply formal in those places. "If you start early enough, you can usually get designers to put in the right assertions," he said.

Why two different tools? Their use models are different and complementary, Ho said. The 0-In tools, he noted, are adept at "handling large numbers of assertions simultaneously very quickly." Mentor also offers a dynamic mode that runs with simulation. JasperGold, in contrast, is purely a static property checking tool. Ho said that JasperGold is "good at allowing the user to guide the analysis for real problem properties. Its fine-grained control over the tool helps us get closure on a lot of properties."

Ho uses 0-In's dynamic mode for occasional "problem cases." A recent example was a possible bug involving two writes to the same read address on a dual-port RAM. But for the most part, Ho uses static checks with both 0-In and JasperGold. "In dynamic mode you're really doing bug hunting," he said. "We lean more and more to getting proofs and conclusive results, and that guides us to the static methodology."

Static proofs, Ho said, uncover problems that are too difficult to find with simulation. An example is an arbitration scheme where one needs to verify all the different masters and clients, taking into account timing differences. "Formal gives us a way to write a property to show there's no violation in the arbitration protocol," he said. Ho noted, however, that formal property checking can replace block-level simulation "only in very rare cases, and only when the block is small and relatively simple."

Ho uses the Accellera Open Verification Library ([OVL](#)) and SVA to write properties. "It's relatively easy to write properties, but it's very difficult to write good properties," he noted. "You can write properties that are great for simulation and horrible for formal verification." Formal specialists at D.E. Shaw, he said, spend a lot of time going through properties and rewriting them to avoid a state-space explosion in the formal tools.

Ho said it's very difficult to predict capacity limits up front. Sometimes a complex property can "blow up" on a simple block, and sometimes local properties can be easily proven on very large blocks. "It's part of our assumption that we will have capacity problems and we will have to modify either the assertion or the way we run it," he said.

Ho said he'd like to see unified coverage between simulation and formal verification. "There should be a method to say, 'we covered that with formal and we don't have to simulate it,'" Ho said. He'd also like to see the tools inform the user when properties need to be rewritten.

Ho's advice to new formal users is to focus on the methodology. "I think a lot of formal tools are pretty comparable," he said. "But if you don't have a good methodology in place, and you're not integrating the results back into your verification plan, it will be very hard to show what the return is on the amount of effort you spend on formal."

### **MIPS: Getting results faster**

[MIPS](#) has been using formal property checking for several years, and currently uses both Cadence's IFV and Synopsys' Magellan, according to Ali Habibi, design and verification engineer. The proof engines for these tools are similar, he said, and MIPS uses both because the company doesn't want to confine itself to just one formal tool.

Formal tools can greatly speed the verification process, according to Habibi. He noted that writing directed and random tests for simulation can take a long time. "With a formal tool, you will be able to get results much faster," he said. "In the beginning, this helps capture all the easy bugs." Later on, he noted, formal tools can help track down more difficult bugs by employing complex proofs. The number of false positives will diminish as the verification progresses, he said.

Habibi said that MIPS engineers start writing properties after they receive an engineering specification. They then try to prove those properties at the block level. If there are 100 properties, Habibi noted, perhaps 80 will be proven, and the remaining 20 can be used for bug hunting. The formal tools, he said, "give you much more confidence, and then with simulation you get more coverage."

While MIPS uses both static and dynamic formal verification, Habibi said he likes the dynamic mode because of its integration with simulation. "You can keep your assertions running in simulation, and even if you don't prove the property, you get better coverage," he said. If you provide the right assumptions, he noted, the formal tool may reduce the need to write random tests. For example, it may find the right sequence of events to overflow a queue – a task that would be challenging with simulation alone.

MIPS engineers use OVL for simple assertions, and use SVA for more complex ones. While writing properties is "pretty straightforward," Habibi noted that the verification engineers who write properties at MIPS typically have at least four years of experience. "You need at least one or two experts who can generate tool-specific properties," he said. It's also important to constrain inputs in order to achieve exhaustive proofs, he noted.

While some complex assertions may not be fully verified, even in a simple block, Habibi said that "80 to 90 percent of the assertions for our blocks are done within 24 hours." Still, he noted, formal tools today are limited to small to medium-sized blocks. He'd like to see formal engines that can handle larger blocks, as well as more advanced constraint solvers.

Habibi's advice to new formal users: "Use it on small blocks first to get confidence in the tool and understand how it works. With more expertise, move to larger designs and integrate formal with simulation. Formal and simulation must go together in the same flow. Don't expect formal to replace simulation, and don't expect simulation to provide a full proof."

### Infineon: Integration with simulation

At Infineon, formal property checking is on an "equal footing" with dynamic verification, according to Blank. Infineon uses 360MV, a static formal property checker from OneSpin Solutions. OneSpin was originally a spinoff from Infineon, and 360MV's predecessor, GateProp, was developed at Siemens' chip division before the division was spun off to become Infineon.

Blank said that property checking is initially used by design engineers to do some early debugging of RTL code. When the RTL is mature, then verification engineers use property checking for module-level verification. It's also become very useful for post-silicon verification and reverse engineering, Blank said, when there's a need to analyze unexpected behavior. "A property checker can give you an answer in a couple of minutes. Simulation will take more time than that to boot up," she said.

360MV uses a proprietary property language called ITL. "Being familiar with ITL, SVA and PSL [Property Specification Language], I consider ITL the better and simpler language for formal verification, but industry standards are important to us," she said. Thus, she noted, Infineon requested the SVA support that [OneSpin recently added](#).

Infineon verification engineers use property checking for both bug hunting and full block-level proofs, Blank said. Full module verification, she said, is left "more or less to the experts. If you want to do full module verification you have to work very carefully."

Infineon has developed a methodology called SimCoForm that helps engineers coordinate formal verification with simulation. The methodology helps the verification engineer determine where to apply formal property checking in order to reduce the coverage-driven verification effort. Even so, Blank noted, "simulation is still our workhorse in verification. There are applications where you just need to see a waveform and a 'property holds' message is not enough."

## SimCoForm - Concerted Verification

### Classification on level of verification objectives

- Assignment of single verification objectives to constrained randomized simulation or formal property checking.

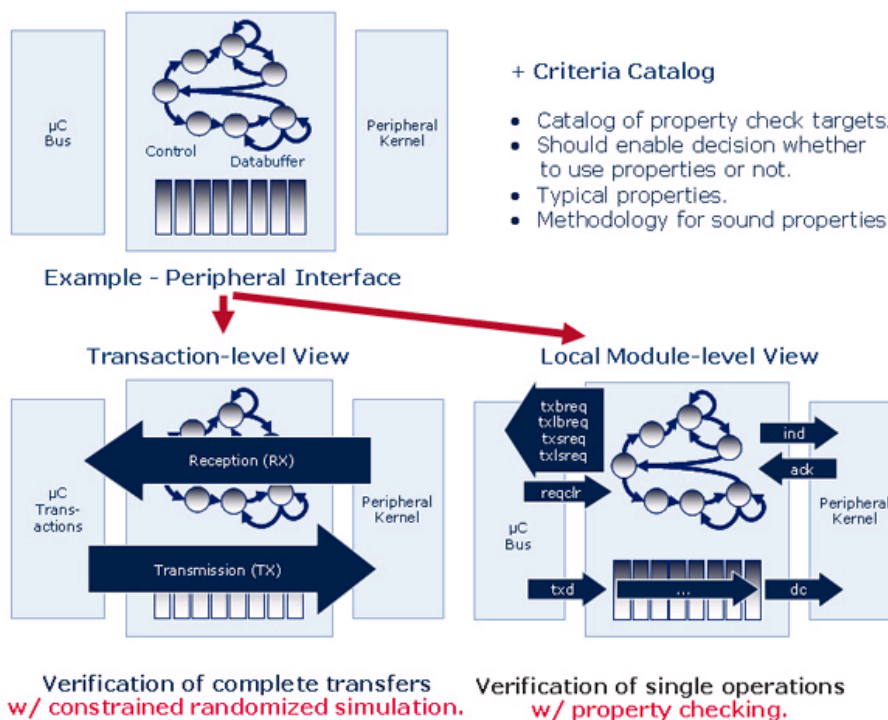


Figure 3: Infineon's SimCoForm methodology combines transaction-level simulation with property checking for single operations. (Source: Infineon Technologies)

Blank said that formal property checking can replace module-level simulation for some blocks such

as arbiters, FIFOs, decoders, and interrupt control units. In most cases, however, Infineon uses both simulation and property checking.

When capacity problems occur with 360MV, Blank said, engineers use a "divide and conquer" approach with the formal proofs. But for most modules, she said, there's no capacity problem. She noted that Infineon verified the TriCore automotive processor using 360MV. This processor has over 100,000 lines of code, and [was verified](#) using 1,500 properties with a regression run-time of seven days on a single workstation, an average of six minutes per property. Infineon and OneSpin [presented a paper](#) about the TriCore verification at the DVCon conference in 2007.

Blank said she would like to see better support for configurable IP in both simulation and formal property checking. As of today, she noted, it's necessary to run a regression test for every possible IP configuration.

Blank's advice for formal verification newcomers: "Just start using it to find bugs. The more experience you get, the more you can move from bug hunting towards systematic formal verification."

### **IBM: Driving designer adoption**

IBM uses formal property checking extensively for its Power processors, according to Viresh Paruthi, technical lead for formal verification at IBM Systems Group. His team uses two formal tools developed by IBM – RuleBase, which is sold externally, and SixthSense, which is used only internally. Both tools perform dynamic and static property checking, and SixthSense additionally performs sequential equivalence checking.

Paruthi heads up a team of formal verification specialists, and part of their mission is to drive wider adoption of formal tools by designers and verification engineers. "In my mind, designers are the best formal verification engineers, because they have such a strong grasp of the micro-architecture and know exactly what they would like to go after," Paruthi said.

At IBM, Paruthi said, design engineers typically use low-level assertions initially to find early bugs. Paruthi's team works with designers to understand the more complex events that need to be verified, and to ensure that formal proofs can verify those events without running into capacity limits. At some point, he said, designs are handed over to formal experts who perform bug hunting and property verification.

Paruthi said that formal verification typically starts with a dynamic approach, and later moves to static property checking. By running with simulation, Paruthi said, a formal tool can help ensure higher coverage, and can investigate signals that would not be driven by simulation alone. The formal tool can apply a test vector and observe expected values at outputs.

Static property checking can find a wide range of bugs, Paruthi said – buffer overflows, liveness bugs, and arbitration bugs, to name a few. It can run end-to-end checks that verify entire protocols. But formal property checking is "complementary and synergistic to block-level simulation," Paruthi said.

To write properties, IBM uses a proprietary language that Paruthi described as an extension to VHDL. Any properties written for formal verification can be used in simulation, he said, but many properties written for simulation cannot be used in formal verification or acceleration. Properties written for simulation don't have to be synthesizable, which is a requirement for formal verification and hardware acceleration, he said.

Paruthi said that RuleBase and SixthSense provide the "best capacity in the industry," and that IBM has verified blocks with millions of gates. He acknowledged, however, that "if we get a design with a lot of activity, we may find it difficult to get the proof. The technology is computationally complex. We have a whole suite of algorithms that we apply when we run into roadblocks."

Paruthi would like to see improved speed and capacity for formal tools. Another item on his wish list is synthesizable testbenches. "If our C/C++ testbenches were synthesizable, they could be reusable in other verification paradigms," he said. A third request is unified coverage between different verification methodologies.

### **James Lee: An automated approach**

What about design teams that don't have access to formal verification specialists, or expertise in writing properties? An automated formal proof tool may be the answer. According to James Lee, author of the [Verilog Quickstart](#) guide to Verilog. Real Intent's EnVision (formerly called Verix) can find bugs with its "implied intent" or automated assertion capability.

Lee used EnVision in the past as a consultant, and he uses it today as design manager at a semiconductor company he declined to identify. "We have used Real Intent on three chips in the last year, and in every case, we found something interesting," he said. Of the various automated checks EnVision runs, Lee thinks that the "constant net check" provides the greatest value. This check finds signals that are stuck at 0 or stuck at 1.

Lee said designers use EnVision early in the verification cycle, "almost too early, because we sometimes forget to rerun it." If a design is "Verix clean" before it goes into simulation, he said, the design team has a lot more confidence. "We really don't do much simulation," Lee said. "We check everything in Verix at the module or subsystem level."

While it's "really, really easy" to use EnVision, there's a tradeoff, Lee said. "Because it is so automatic, the number of cycles and the amount of depth it can explore in the circuit is somewhat limited." A common problem, he said, is a state machine that remains in a given state until a counter expires. "I wish they would do an intelligent extraction and say, 'that's a counter, let's apply an algorithm that tests counters.'"

Lee said that "formal tools are really valuable. There's a lot of information you can get out of them." With many tools, however, "you have to become an expert in writing properties. That's why I like Real Intent – you get so much for so little."

## Conclusions

It's clear from user experience that formal property checking tools require some expertise. They don't necessarily require PhDs in formal verification, but to get the best use out of the tools, someone has to know the tool well, understand how to write good formal properties, and know what to do when proofs don't complete. Other significant points:

- Formal tools complement, rather than replace, module-level simulation.
- Dynamic tools don't offer complete proofs, but they can handle larger blocks than purely static tools, and can be a valuable adjunct to simulation.
- The capacity of static formal tools is increasing, but more progress is needed.
- Post-silicon debug is becoming an attractive application of formal property checking.
- Tools with automated checks have limitations, but are easy for designers to adopt and use.

The most significant conclusion, however, is that both static and dynamic formal tools have real value, and are an important part of the verification methodologies deployed by the users profiled above.

The second part of this report will discuss capabilities of some of the tools, and provide vendor viewpoints on the applications, limitations, and future prospects for formal property checking and "bug hunting" tools.

## Related articles

[Verification coverage measures up to higher level](#)

[Jasper eases formal proofs, updates planning tool](#)

[Formal tool opens 'gateway' to assertion reuse](#)

[How assertions improve design and verification](#)

## Reference

[Complete Formal Verification of TriCore2 and Other Processors](#)



Add Comment - please [log-in](#) to comment

SCDsource newsletter subscribers may post a comment - [Register for free!](#)

[Back to Home Page](#)