

Contributed Article

[Print](#) [Email](#) [Bookmark](#)

Using formal verification for post-silicon debug

By Lawrence Loh and Jay Littlefield, Jasper Design Automation

04/23/08

The burden that system-on-chip (SoC) design complexity places on logic designers and verification engineers is well-documented. But what about the silicon bring-up team? What happens when a critical bug slips through to silicon? Further costly design re-spins must be avoided, so it is absolutely essential to thoroughly debug the silicon as quickly as possible. But that's very difficult because of the limited visibility into silicon.

Debugging silicon can be a very expensive process. At a DesignCon 2006 panel, MIPS CEO John Bourgoin stated that "finding bugs in model testing is the least expensive and most desired approach, but the cost of a bug goes up 10X if it's detected in component test, 10X more if it's discovered in system test, and 10X more if it's discovered in the field, leading to a failure, a recall, or damage to a customer's reputation."

Despite these costs, there are very few EDA tools available to tackle post-silicon bugs. However, formal methods are now cracking the problem because of their ability to resolve verification ambiguity. Because the properties used in post-silicon debug are complex, high level properties, the chosen formal verification tool must be able to handle the associated capacity issues. That precludes the use of formal assertion-based verification (ABV) tools because their assertions operate at too low a level. So, how do we go about it? First, let's understand more about what causes bugs to be missed.

Finding bugs in silicon

Ambiguity in post-silicon debug is a critical barrier. Verification ambiguity includes incomplete testbench setups, errors and omissions in the property specification, software bugs, and so on. Each of these factors is a potential bug source, so eliminating them from consideration early in the process is key to isolating the root cause of the bug.

Simulation, emulation and other vector-based approaches can never really remove a potential source of the problem from consideration because there are always new, yet-to-be-run vector sets that might stimulate a problem in any of these areas. Formal verification eliminates the ambiguity. Because of formal's exhaustive analysis of behavior, it conclusively determines whether or not a particular verification component is the source of the undesired behavior. Formal, therefore, quickly isolates the root cause of the problem. Once identified, the team can determine whether to fix the bug in software or to do a mask respin.

Normally, when a bug is found in silicon, additional simulation tests are run to try to identify the source of the undesired behavior. This may consist of additional constrained-random simulation to increase bug coverage, as well as directed tests that are plausible triggers for the bug. The existing system-level testbench is most often used since there is little time available to create a custom silicon debugging environment.

The problem is that the system level testbench operates at such a high abstraction level that the necessary level of control for the internal signals of interest may not be possible. Testing specific scenarios of interest at this level of the design is therefore quite difficult. Additionally, the more lengthy and specific the required trace is to trigger the bug, the less likely it is to be covered using a constrained random simulation environment.

Lastly, one second of actual run time in the lab is typically equivalent to several hours of full-chip simulation time. Identifying difficult bugs that occur only after days or weeks of actual silicon run time is extremely challenging with simulation, particularly if there are few hints as to what operations cause the failure. With vector-based approaches such as these, you never know quite how far away you are from discovering the cause of the bug, and the only solution is to keep testing additional vectors in the hopes that you will stumble across it.

Formal verification resolves the signal controllability issue by ignoring the testbench altogether. Formal verifies a design by exhaustively testing its behavior against a specified property which should always hold true. If a violation is found, a waveform is created to show how the property fails. Because there is no testbench scaffolding surrounding the design, formal can be run at whatever level of design makes the most sense for the verification effort.

The input stimuli for the analysis are generated by the formal tool itself, and not by an external source. All possible input scenarios are determined algorithmically by the formal tool for full controllability and complete state space analysis of the design-under-test. This is why formal analysis is capable of generating conclusive answers to verification problems. The post-silicon formal debug flow is shown in Figure 1.

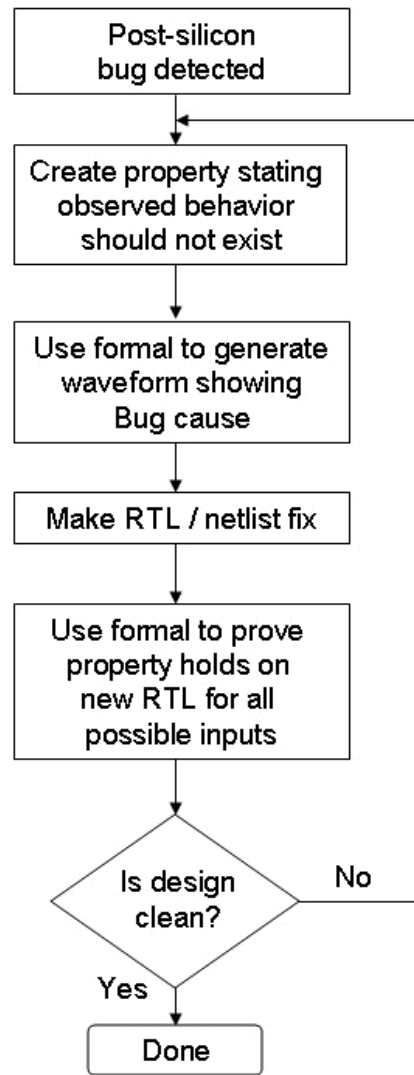



Figure 1 – Post-silicon formal debugging flow

To identify the source of the post-silicon bug using formal verification, all that is needed is a property declaring that the observed bad behavior does not exist. Since the behavior is known to exist within the design, formal verification should fail the property. When it does, the formal tool should generate a waveform describing the exact sequence of events to reproduce the problem.

Since formal verification tests all possible input sequences, the analysis is not dependent upon guessing the correct vector to stimulate the bug. Formal provides the path for you. With an appropriate property, formal verification can find the bug in a mere fraction of the time it would normally take to diagnose the very same bug in the lab or using emulation. In many cases, this time is weeks to months shorter than alternative solutions.

 [Next Page](#)

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

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

[Back to Home Page](#)