cadence®

Formal Verification for Post-Silicon Debug

Bug escape costs grow considerably with each and every subsequent step in the design flow, to the point of being exorbitantly high once at the silicon level. As a result, these high costs of bug escape are driving customers to embrace formal verification for post-silicon debug and to begin using formal far earlier in the flow for their next design projects. The Cadence® JasperGold® Verification System's ability to quickly identify bugs, assure the cleanliness of sub-blocks, and verify the completeness of design fixes makes it the highest value post-silicon debug tool in the team's arsenal.

Contents

Introduction

It is a well-established and supported belief that today's ever-increasing SoC design complexity is placing an enormous verification burden on logic designers and verification engineers. But what about the silicon bring-up team? While verification teams work hard to prevent such issues, few problems can cause a development team as much heartache as a critical bug that has slipped through to silicon. This is due to the fact that such bugs have the potential to cause a disastrous turn of events in the development cycle of a chip. Once in silicon, a critical bug can lead to hugely costly design re-spins, lost market opportunities, or even complete business failure. Therefore, it is absolutely essential to resolve the issue as quickly as possible. And yet, there are very few EDA tools able to tackle post-silicon bugs due to the highly reduced visibility of silicon.

Mounting market pressures are now spurring a growing interest in the use of formal verification for post-silicon debug. The reason is simple. Once a design is in silicon, large sums of money are on the line. With its proven ability to remove verification ambiguity, formal verification offers the greatest promise for conclusive answers for the tremendous challenge associated with post-silicon debug. Used properly, formal has the ability to significantly reduce both the time and cost of post-silicon debug. The resulting faster turnaround enables teams to still meet their tight market windows.

While formal verification was not considered for post-silicon debug in the past, today it has been proven to be ideally suited to this task. Naturally, the chosen formal verification tool must have the production-proven ability to handle the capacity issues that will accompany any post-silicon debugging effort, since properties modeled on observable behavior will be high-level, complex properties. Since formal assertion-based verification (ABV) tools use simple, low-level assertions, they would be unsuitable for post-silicon debug. In the course of this article it will be detailed why deep formal is now moving to the mainstream for post-silicon debug.

Finding Bugs in Silicon

Once a design has achieved silicon, debugging that design can be a very expensive process. At a DesignCon 2006 panel, then-CEO of MIPS, John Bourgoin, stated, "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." This relationship is shown in Figure 1. Clearly, debugging errors in post-silicon, at the end of the design cycle, incurs great expense.

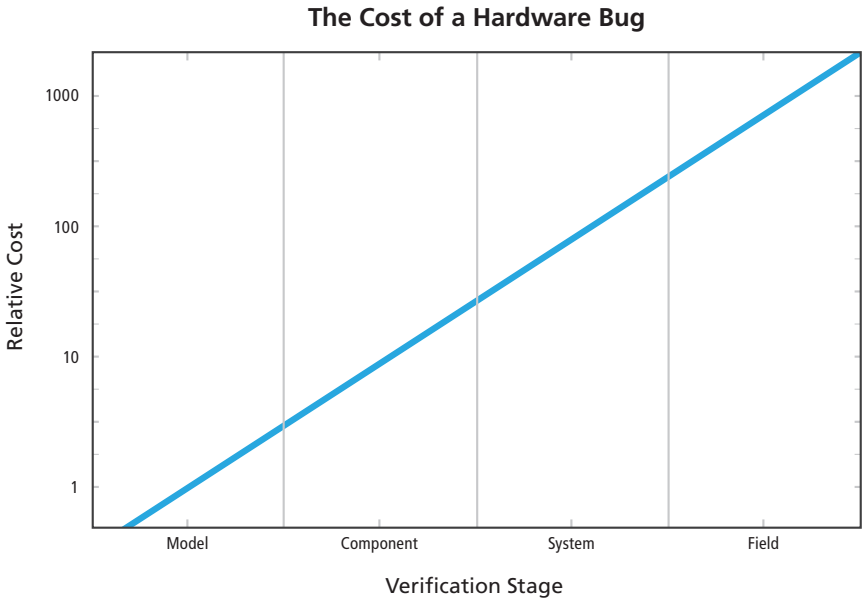

Figure 1: The cost of a hardware bug

As previously mentioned, ambiguity in the post-silicon debug process is a critical factor. Verification ambiguity includes incomplete testbench setup, improper property specification, software bugs, etc. 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 any potential source of the problem from the equation because there are always new vector sets that haven't been run and might stimulate the problem in any of these areas. The benefit of deploying formal verification in post-silicon debug is to remove ambiguity from the process. Because of formal's exhaustive analysis behavior, conclusive answers can be generated as to whether or not a particular verification component is the source of the undesired behavior. Formal therefore enables quick isolation of the root cause of the problem. Once identified, the team can determine if the bug can be dealt with using a software fix, or if a mask re-spin is required.

Additional Bug Tracing Issues

Traditionally, when a bug is found in silicon, additional simulation tests are run to try and 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 post-silicon debugging environment. The problem is that the system-level testbench exists at such a high design 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. Finally, one second's worth of actual run time in the lab is typically several hours' worth 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 that 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 (DUT). This is why formal analysis is capable of generating conclusive answers to verification problems.

To identify the source of the post-silicon bug using formal, all that is needed is a property declaring 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 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.

For any formal analysis, proper attention must be paid to the capacity of the problem to be verified. Most formal verification solutions in the market today focus on low-level ABV-style properties, verifying localized properties with small input cones of logic. These types of formal tools are poorly suited to handling the large capacity of the debugging problems involved in post-silicon debug. Only a formal tool with a focus on full proofs of complex design properties will have the capacity and methodology to effectively converge on the problem within the time frames required to meet the demanding market window timelines (see Figure 2). The tool most companies are using today for this purpose is the JasperGold Verification System. It not only has the ability to identify deeply embedded bugs quickly, but also can prove that the bug fixes to remove them are complete and have no other unintended side effects that may also violate the property being tested. The JasperGold Verification System is used regularly in production to identify issues several hundreds of clock cycles long on block-level, end-to-end, and high-level properties. This unique ability makes it ideally suited for discovering the root cause of many deeply hidden post-silicon debug problems.

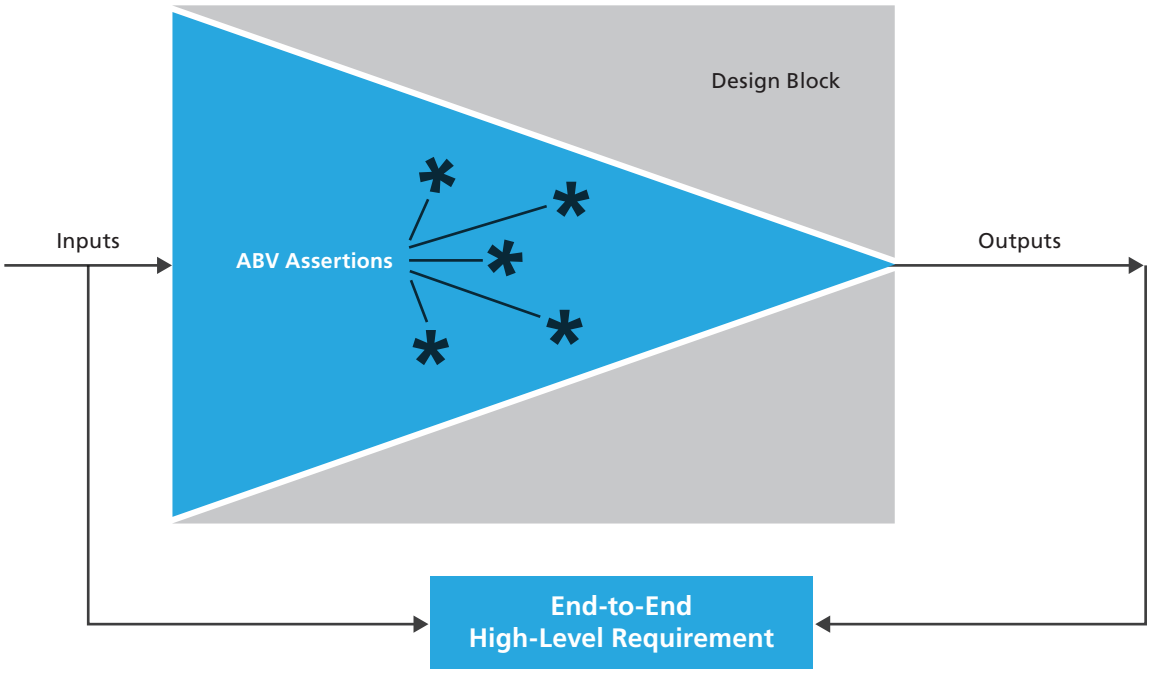

Figure 2: ABV vs. high-level properties

Proving the Absence of a Bug

Proving that a bug fix completely addresses the silicon problem is a crucial part of any post-silicon debug effort, and its importance cannot be overstated. On many occasions, verification teams have discovered only after another re-spin that the lab fix was NOT complete, with a deeper and subtler bug symptom still present. Formal verification can save the re-spin cost by identifying additional bugs once a fix has been made to the initial problem. Verifying

this type of fix with simulation or emulation is a hit-or-miss proposition, because success still depends upon finding exactly the correct sequence of input behavior to trigger the bug. As the bug symptoms are pushed deeper into the design, they often involve longer and longer sequences to trigger them. The effectiveness of vector-based verification drops dramatically when the target is an unknown yet specific long sequence of events. For formal, it's simply a matter of running deeper analysis while handling the additional capacity and complexity.

Dealing with Complexity

Even in the case where the problem under analysis is too large for formal verification to converge completely, it can still be quite helpful in the post-silicon debug effort. More often than not, a high-level property can be easily decomposed into a handful of block-level end-to-end properties where formal can run effectively. At one customer site, intense effort was underway to identify the source of a post-silicon bug within a large network-processing chip. The debugging effort was focused on a single sub-block within the design. A top-level property describing the symptom was too complex to run at the system level, and was quickly decomposed into individual block properties. The JasperGold Verification System quickly proved that the block under inspection was actually clean with respect to the bug in question. Further formal analysis identified that the downstream components from this block were also clean. With this knowledge, attention turned to blocks earlier in the design flow, where the source of the bug was quickly identified and fixed. Without formal verification, much more time and effort would have been spent on the downstream blocks, with the real possibility of missing the target market window.

Conclusion

Formal verification for post-silicon debug is a fairly recent phenomenon—only over the last few years have the market-leading electronics companies been quietly applying such an approach to great success. To avoid the higher associated costs of post-silicon bug escapes, companies should consider working with formal verification earlier in the design flow, starting at the microarchitecture-level where the actual cost of catching a bug is very low. If companies make it a regular habit to catch any bugs at this level, they can potentially avoid any disasters due to downstream bug escapes. Unfortunately, bug escape costs grow considerably with each and every subsequent step in the design flow, to the point of being exorbitantly high once at the silicon level. As a result, these high costs of bug escape are driving customers to embrace formal verification for post-silicon debug and to begin using formal far earlier in the flow for their next design projects.

With formal's unique ability to remove verification ambiguity, it becomes an invaluable tool in reducing the time and effort spent addressing post-silicon debug issues. Formal's ability to quickly identify bugs, assure the cleanliness of sub-blocks, and verify the completeness of design fixes provides the highest value post-silicon debug tool in the team's arsenal. Of course, the proper verification tools, high-capacity formal engines, and a proven deep formal methodology are all essential in ensuring success with formal verification. The Cadence JasperGold Verification System is uniquely positioned to provide end users with high-value, high-throughput post-silicon debugging.

Further Information

To learn more about Cadence JasperGold Apps, contact your local sales office at http://www.cadence.com/cadence/contact_us.