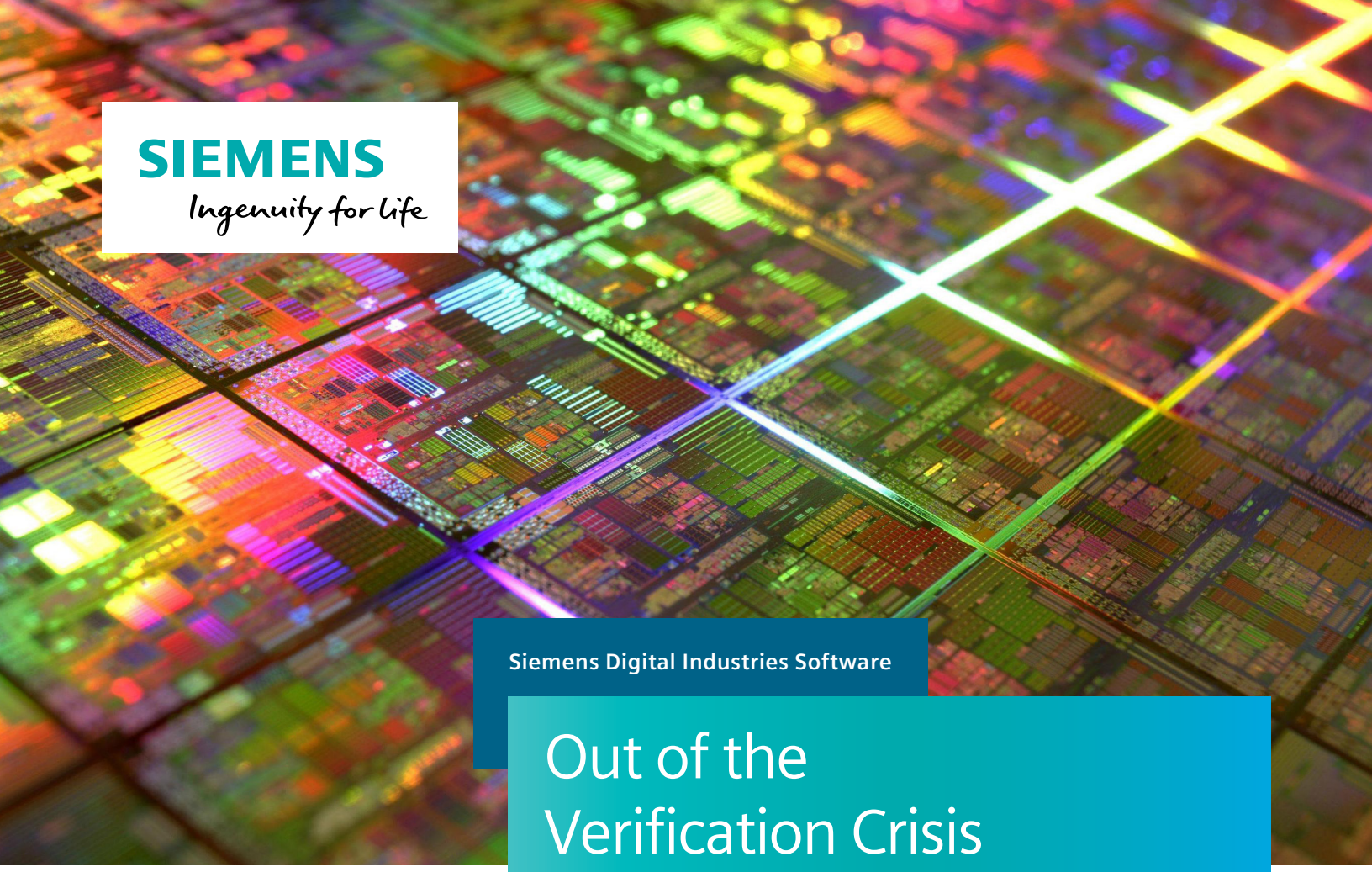

SIEMENS
Ingenuity for life

Siemens Digital Industries Software

Out of the Verification Crisis

Improving RTL Quality

Executive summary

A verification crisis is upon us that will not be solved solely through improvements in verification methodologies and techniques. The solution requires a holistic and philosophical change in the way we approach design with a foundation based on bug prevention. Our proposed first step in implementing this change tightly integrates static analysis into the design process, resulting in a decrease in bug density, which has a positive impact on downstream processes and consequently reduces cost.

Harry Foster – Siemens Digital Industries Software

The Crisis

In 1997, SEMATECH set off an alarm in the industry when it warned that IC manufacturing productivity gains were increasing at a 40% CAGR, while IC design productivity gains increased at only a 20% CAGR. This concern was reiterated in the International Technology Roadmap for Semiconductors 1999 report [1]. Despite these alarms concerning the gap between silicon capacity and design capabilities, the industry avoided this crisis. Why? There were two primary contributors that prevented the design productivity gap: (1) continual improvements in design automation and (2) the emergence of a silicon IP economy that fueled a productive design reuse strategy [2].

In the last decade, a more ominous productivity gap has emerged with respect to verification. While silicon complexity grows at the Moore's Law rate, verification complexity grows at a significantly greater rate , and the approaches that were used to close the design productivity gap will be insufficient in closing the verification productivity gap. IBS [3] quantified the impact of today's verification gap in terms of IC project's verification and validation cost with respect to decreasing process node feature size, as shown in Fig. 1.

Additional industry studies have measured the verification productivity gap's impact on IC projects, such as the 2020 Wilson Research Group functional verification study [4]. For example, since 2007, the mean peak number of design engineers working on a project has increased by 32%, while the mean peak number of verification engineers has increased by an alarming 143%. In fact, today there are more verification engineers on average working on an ASIC/IC project than design engineers. Yet even with the increase in project headcount, 66% of all ASIC/IC projects experience one or more respins, while 83% of FPGA projects experience one or more non-trivial bug escapes into production [5]. In addition, two thirds of all ASIC/IC and FPGA projects miss their originally planned schedule. Clearly, a verification crisis is upon us.

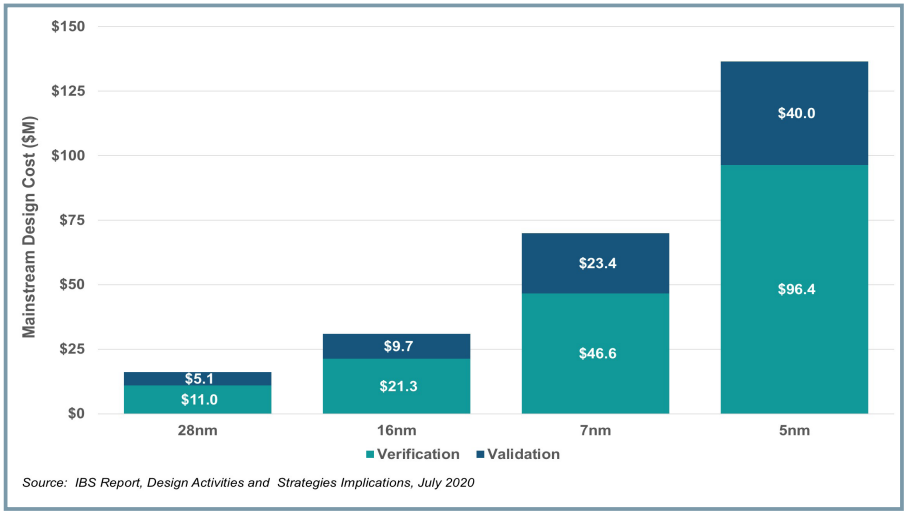

Fig. 1. IC verification and validation cost by process node feature size.

The Problem

In the 1990s, many projects began to organize into separate design and verification teams for two reasons: (1) to ensure an independent team interpretation of the specification that would assist in flushing out design errors and (2) the complexity of verification environments increased and required unique engineering skills to create them. While this project organizational change had a positive impact on identifying bugs associated with a misinterpretation of the specification, it has also led to a fallacy that quality can be verified into a product, and that the verification team is exclusively responsible for functional quality.

> Quality cannot be inspected into a product; it must be built into it.

W. Edwards Deming in his landmark book Out of the Crisis revealed that "quality cannot be inspected into a product; it must be built into it [6]." Deming was famous for conducting exercises in his quality management training courses. To illustrate his point about quality, he filled two wooden containers with white beads and then added a handful of red beads to represent bugs. He then divided his class into two groups and gave each group one of the wooden containers with a paddle to remove the red

1 *In theory, verification complexity grows at a double exponential rate; but in practice, it grows closer to a quadradic rate, which is a significantly greater rate than the growth of design complexity.*

beads. His professed plan was to reward the group that removed the greatest number of red beads and penalize the other group. While there were numerous process management lessons learned through this exercise, a profound outcome was uncovered. After the exercise, each group shared their experiences and chiefly focused on the techniques they developed to remove the red beads. Deming, however, told the class that they missed the point. It didn't matter if you rewarded or punished a group in terms of how effectively they removed the red beads: the results were, in all practicality, the same because some red beads remained. His profound takeaway was that finding an optimal process to remove the red beads was not the solution. The real solution was to not put the red beads in the wooden container in the first place. This wisdom is applicable to IC design today.

Bug prevention must become the foundation of the design process.

An industry analysis of IC-respin root causes has revealed cases where not only complex corner case bugs are escaping into production, but also trivial bugs. For example, even after billions of cycles of coverage-driven, constrained-random simulation, combined with directed testing, one high-end server project in the analysis experienced an "out-of-range indexing" error that resulted in data corruption. This simple, yet costly respin could have been avoided if the design engineer simply ran *lint* on the RTL code prior to checking it in. Without question, this outlier is an extreme case. Yet even if this simple RTL bug was caught prior to tape out, the cost of triaging, debugging, and redesigning, followed by additional verification is significantly higher when caught later in the development lifecycle.

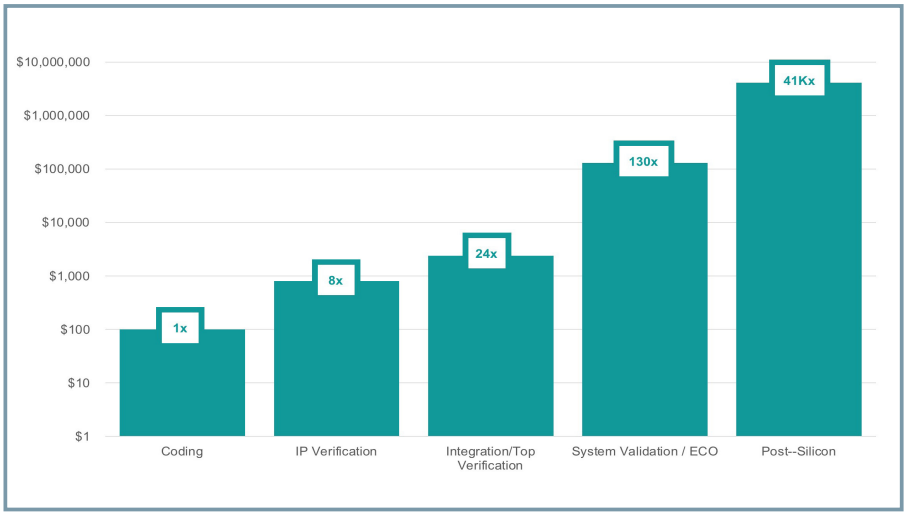

Fig. 2. Cost of finding and fixing a bug at various development stages for a 5 nm ASIC.

To help understand the cost impact on a project, in Fig. 2 we quantify the cost multipliers associated with finding and fixing bugs at various stages in the development lifecycle. In earlier stages of a project, the cost is predominately due to labor and other resource expenses. Notice that a bug found at the IP verification stage is 8x more costly than if found at the coding stage. This is due to the increase in resources required to identify and triage bugs after coding. Obviously, the cost multiplier increases significantly during the post-silicon stage where expenses include not only labor, but also expenses associated with building hardware, such as silicon, validation boards, or prototypes.

The Prescription

Design bugs are introduced throughout all phases of the development lifecycle, including the architectural, design, synthesis, integration, and physical design phases. To reduce the cost of finding and fixing bugs, methodologies that promote testing early and often have recently emerged, such as shifting the verification phase to earlier in the development lifecycle combined with continuous integration.

Today's crisis will not be solved solely through improvements in verification methodologies and techniques. A solution requires a holistic and philosophical change in the design process with a foundation based on bug prevention. We refer to this fundamental change as *design using intent-focused insight*, or *design+intent*.

A shift-left development lifecycle that incorporates design+intent does not make verification any less important than it has been. In fact, the goal of a design+intent process is to improve verification efficiency by decreasing bug density, which impacts downstream processes, and consequently reduce cost. While a design+intent bug prevention strategy encompasses all aspects of IC design, there are design solutions that exist today with a principal focus on improving RTL quality, such as static analysis.

Static analysis is a non-simulation-based testing activity in which the RTL code is analyzed for defects ranging from non-compliance with the specification to those known to be associated with design bugs. Static analysis can also be used to find incorrect transformations as the design progresses through various implementation phases. Fully automated static analysis solutions range from lightweight tools, such as lint, to advanced bug-hunting static-analysis tools that use formal technology. The value in adopting static analysis solutions is a significant improvement in verification debugging efficiency due to the reduction of bugs during RTL handoff. Static analysis is an easy to adopt first step of a project's overall design+intent bug prevention strategy.

The Solution

A design+intent solution is built on three functional pillars, as shown in Fig. 3.

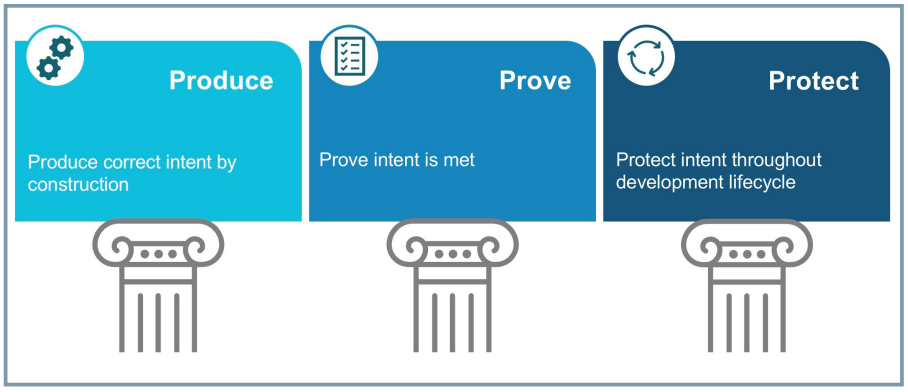

Fig. 3. Three pillars of a design+intent methodology.

The first pillar, labeled *Produce*, consists of a process that produces the correct design intent during construction to minimize bugs. The second pillar, *Prove*, ensures that the designer's intent and requirements are met early in the development lifecycle. And the third pillar, *Protect*, ensures that the design intent is preserved as the design progresses through the various stages of the development lifecycle.

Before we discuss each pillar in the design+intent solution, we should examine the origin and nature of bugs. One question many engineers ask is if certain RTL languages are more susceptible to bugs than others. The answer is no. In fact, this has been observed across many software projects for years, where the specific choice of language was irrelevant in terms of number of bugs. Indeed, this has been quantified across multiple software projects in terms of number of bugs per 1000 lines of code (LOC). On average, software projects consistently observe between 15–50 bugs per 1K LOC, depending on the complexity of the code. And this software bug density pattern is also true for hardware projects using RTL.

Another question many engineers ask is if certain design application domains are more susceptible to bugs than others. The answer is not necessarily. What has been observed on projects is that design blocks that are concurrent in nature, with multiple concurrent data streams, contain 5x the number of bugs versus design blocks that are sequential in nature, as shown in Fig. 4. For example, in general, a new DMA controller or a new PCIe block will likely experience 5x more bugs than a new DSP convolution unit or a MPEG decoder block. This is due to corner-case bugs

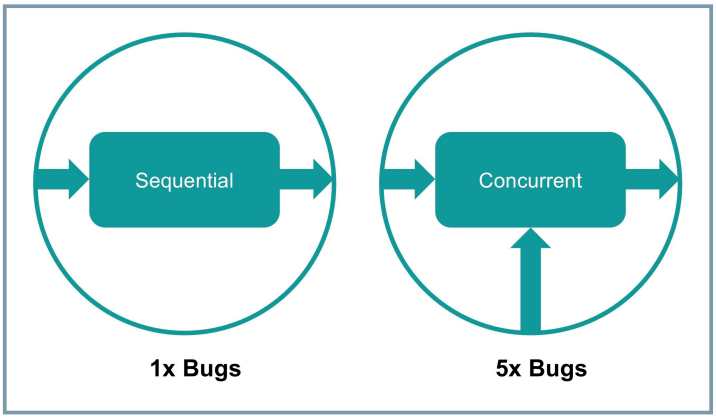

Fig. 4. Bug density by design style.

often associated with concurrency. The good news is that blocks that are concurrent in nature are generally better suited for formal techniques.

Now that we understand the nature of bugs, we can explore how a design+intent methodology can help identify and fix issues when the cost multiplier is small. We begin by exploring various solutions for the Produce pillar, and the first solution could be to raise the level of design abstraction while leveraging high-level synthesis whenever possible. Why? By leveraging a higher-level language (HLL), such as C/C++, we reduce the number of lines of code that are required to describe the design. For example, in many cases, 100 lines of HLL is equivalent to 1000 lines of RTL. With this reduction in the number of lines of code, we should also expect a 10x reduction in the average number of bugs. For our example, we would expect 1–5 bugs for the HLL design versus 15–50 bugs for the equivalent RTL design.

Yet not all design blocks lend themselves to high-level synthesis. Hence, another key part of the Produce pillar is an HDL design environment that integrates deep analysis capabilities into the creation process. These can quickly assess new and reused code quality to prevent bugs during development.

The Prove pillar is the core of a design+intent solution. It provides the insight that ensures the designer's intent is met. The analysis performed in the Prove pillar falls into two major categories. The first category involves RTL code syntactic, semantic, stylistic, and structural analyses, which identify coding or methodology errors that are costlier to find and fix after the code is checked into the regression.

The second category involves sequential analysis, which leverages advanced bug-hunting static-analysis and formal technology. By employing sequential analysis within the Prove pillar, engineers can identify complex corner-case bugs associated with concurrency, as we previously discussed. A few examples of bugs found using sequential analysis include combinatorial loops, FSM deadlocks, arithmetic overflow, and indexing issues. The key point is that by leveraging sequential analysis design solutions these bugs can be found and fixed during the coding stage without the need to create a simulation testbench.

One critical analysis that must be performed within the Prove pillar identifies a class of bugs associated with clocking and reset metastability issues. Indeed, many engineers fail to understand that metastability bugs cannot be demonstrated on an RTL model using simulation and are often found at a higher cost multiplier if not prevented during the design stage. Furthermore, this class of errors is extremely difficult to identify and reproduce in the lab due to their random occurrence.

Another important analysis performed within the Prove pillar identifies a class of bugs associated with RTL X pessimism and X optimism. While X pessimism errors are frustrating and time consuming to identify and fix, X optimism bugs are insidious in that they can mask serious functional errors in the RTL model, particularly errors that result in simulation differences between the RTL and gate-level models that cannot be found using traditional equivalence checking tools. Ideally, these errors should be remedied using static analysis before RTL code is checked into the simulation regression.

Finally, the Protect pillar consists of analysis tools that ensure the intent of the design is retained throughout the entire development life cycle; for example, identifying new metastability issues potentially introduced during the synthesis and implementation process.

One recommendation when adopting a design+intent methodology is to automatically build these analyses into a continuous integration flow, which ensures that the design is protected from faulty changes when moving from creation to completion. This is easily accomplished since the automatic static analysis tools contained in the various pillars generally involve simple-to-no constraints and do not require manual interactions to operate. Indeed, it is possible to implement a set of light high-value checks as a gatekeeper to any regression check-in. This can be followed by deeper analyses for daily and weekend regression runs, while the deepest checks can be performed prior to committing the design to the more intensive prototyping and emulation stages that are often used for hardware/software co-design and system validation.

Summary

Finding a path out of the verification crisis requires a philosophical change throughout the development lifecycle with a foundation built on bug prevention. To begin this journey, we propose that projects focus on design+intent solutions, such as static analysis, that promote more consistent development cycles and faster verification convergence by improving RTL quality.

References

[1] Semiconductor Industry Association. International Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: International Sematech, 1999.

[2] H. Foster, „Why the design productivity gap never happened" in Proceedings of the International Conference on Computer-Aided Design (ICCAD), IEEE Press pp. 581-584, San Jose, California, Nov. 2013.

[3] IBS, Global Semiconductor Industry Service Report, Design Activities and Strategies Implications, July 2020.

[4] H. Foster (2021), 2020 Wilson Research Group Functional Verification Study: IC/ASIC functional verification trend report [Siemens Digital Industry Software white paper]. Retrieved from here.

[5] H. Foster (2021), 2020 Wilson Research Group Functional Verification Study: FPGA functional verification trend report [Siemens Digital Industry Software white paper]. Retrieved from here.

[6] W. Edwards Deming, Out of the Crisis, MIT Press, Reissue ed. edition, 2018.

Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Xcelerator, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

siemens.com/eda

83787-C3 9/21 TGB