Assertion-Based Verification: Industry Myths to Realities
(Invited Tutorial)

Harry Foster

Mentor Graphics Corporation
Plano, Texas
Harry_Foster@mentor.com

Abstract. Debugging, on average, has grown to consume more than 60% of today's ASIC and SoC verification effort. Clearly, this is a topic the industry must address, and some organizations have done just that. Those that have adopted an assertion-based verification (ABV) methodology have seen significant reduction in simulation debugging time (as much as 50% [1]) due to improved observability. Furthermore, organizations that have embraced an ABV methodology are able to take advantage of more advanced verification techniques, such as formal verification, thus improving their overall verification quality and results. Nonetheless, even with multiple published industry case studies from various early adopters—each touting the benefits of applying ABV—the industry as a whole has resisted adopting assertion-based techniques. This tutorial provides an industry survey of today's ABV landscape, ranging from myths to realities. Emerging challenges and possible research opportunities are discussed. The following extended abstract provides a reference on which the tutorial builds.

Keywords: Assertion, Assertion-Based Verification, Debugging, Formal Verification, Functional Verification, Property Specification, Simulation.

1 Introduction

Ensuring functional correctness on RTL designs continues to pose one of the greatest challenges for today's ASIC and SoC design teams. Very few project managers would disagree with this statement. In fact, an often cited 2004 industry study by Collett International Research revealed that 35 percent of the total ASIC development effort was spent in verification [2]. In 2008, Far West Research published a study that indicated the verification effort has risen to 46 percent of the total ASIC development effort [3]. Furthermore, these industry studies reveal that debugging is the fastest-growing component of the verification effort, and that it consumes 60 percent of the total verification effort. Unfortunately, with this increase in verification effort, the industry has not experienced a measurable increase in quality of results. For example, the Collett International Research study focused on design closure and revealed that only 29 percent of projects developing ASICs were able to achieve first silicon

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 5–10, 2008.

success. To make matters worse, the industry is witnessing increasing pressure to shorten the overall ASIC and SoC development cycle. Clearly, new design and verification techniques, combined with a focus on maturing functional verification process capabilities within an organization (and the industry as a whole), are required. Assertion-based verification (ABV), although certainly not an end-all to the verification challenge, does directly address today's debugging problem, while providing an integration path for more advanced forms of verification into the design flow (such as formal verification). This tutorial discussion provides a survey of today's ABV landscape, ranging from assertion language standardization efforts to industry case-studies, to common industry myths and objections that are impeding adoption, to emerging challenges and research opportunities.

2 Background

Alan Turing made the following observation over 50 years ago [4]: "How can one check a large routine in the sense of making sure that it's right? In order that the man who checks may not have too difficult a task, the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily flows." In essence, this view is at the heart of ABV.

Informally, an assertion is a statement of design intent that can be used to specify design behavior. Assertions may specify internal design behaviors (such as a specific FIFO structure) or external design behavior (such as protocol rules or even higher-level, end-to-end behavior that spans multiple design blocks). One key characteristic of assertions is that they allow the user to specify *what* the design is supposed to do at a high level of abstraction, without having to describe the details of *how* the design intent is to be implemented. Thus, this abstract view of the design intent is ideal for the verification process—whether we are specifying high-level requirements or lower-level internal design behavior by means of assertions.

As a background for the ABV discussion, this tutorial traces events (from an industry perspective) that led to the emergence of assertion-based techniques.

3 The Road to Assertion Language Standards

Assertions are certainly not a new phenomenon in either software programming or hardware description languages. For example, languages such as Java and VHDL have contained simple assertion constructs for years —thus providing a convenient mechanism for ferreting out a class of bugs that can be identified by checking a Boolean condition (such as referencing a NULL pointer). Today's assertion languages, such as the IEEE Std 1850™-2005 Standard for Property Specification Language (PSL) and the assertion language contained within the IEEE Std 1800™-2005 SystemVerilog Verification and Hardware Description Language (SVA), not only allow the user to specify Boolean conditions, but also their relation over time using temporal logic and a generalized form of regular expressions.

The foundation for today's assertion language standards is built on the works of Amir Pnueli (linear time logic LTL [5]), and Ed Clarke and Allen Emerson (computation tree logic CTL [6]). Furthermore, the works of Moshe Vardi and Pierre Wolper

provided significant contributions in that they helped improve expressiveness of LTL through the use of regular sequences of Boolean events [7, 8, 9]. Extending the expressiveness of CTL was later demonstrated by [10].

In the early 1990's, researchers at the IBM Haifa Research Laboratory developed the temporal language Sugar, which was a syntactic simplification (or sugaring) of CTL. The goal was to simplify the specification process for the RuleBase model checker. To improve usability and expressiveness, regular expressions were added to the language in the mid 1990's [11]. By the late 1990's, IBM had expanded its use of the Sugar assertion language for simulation [1].

With a similar motive, researchers at Intel Strategic CAD Labs developed the ForSpec property specification language, whose underlying logic is the ForSpec Temporal Logic (FTL) [12], which is based on LTL. Their decision to base FTL on LTL was driven by a desire to combine formal verification and dynamic validation techniques in a limited fashion. Furthermore, experience had demonstrated that mainstream verification engineers generally find branching time unintuitive—particularly since they are familiar with dynamic validation, which is inherently linear.

In 2000, both Sugar and ForSpec, in addition to the temporal property languages CBV from Motorola and Temporal ***e*** from Verisity, were donated to Accellera Formal Verification Technical Committee (FVTC) as candidate languages for standardization. The process within the committee was to establish a set of requirements for an assertion language and select a single language from four candidates. The final selection would then form the basis for the new standard, which then would undergo modification and enhancements dictated by the language requirements identified by the committee. For example, one of the committee's identified requirements was that its underlying semantics for the final standard should be based on linear time. This requirement influenced the IBM team to move Sugar from its branching-time semantics based on CTL to the linear-time semantics of LTL. In 2002, the FVTC selected Sugar as the base language, and it was approved by Accellera in 2004. Ultimately, the IEEE 1850™-2005 Property Specification Language PSL standard, based on the Accellera standard, was approved in October 2005 [13].

In 2002, work was underway in Accellera for the creation of a new version of Verilog, which would combine hardware description and hardware verification language capabilities into a single language. This effort resulted in the IEEE 1800™-2005 SystemVerilog – Unified Hardware Design, Specification, and Verification Language standard, which was approved in November 2005. A major feature of this new language was the addition of temporal assertions, referred to as SystemVerilog Assertions (SVA). SVA has its roots in Open Vera Assertions (from Synopsys), ForSpec, and PSL. SVA provides direct links to control the verification environment by using action blocks associated with its cover and assertion directives. This capability allows the user to create reusable verification IP that can easily communicate with other verification components within the testbench, thus providing a separation between verification IP detection and action. In addition, the language provides a convenient mechanism for expressing a data integrity class of properties through the use of local variables. An SVA local variable provides the benefit of sampling and manipulating data in a property or sequence without requiring the property writer to define auxiliary state machines to model the intended behavior [14].

This tutorial compares and contrasts these two new industry standards, PSL and SVA, and then discusses future language directions for both.

3 Industry Challenges

A few industry surveys indicate that approximately 60 percent of the industry is currently employing assertion-based techniques [15]. However, these surveys are flawed in that they were conducted at conferences with a large attendance of engineers already using advanced verification techniques. From my own experience of engaging with a larger more diverse population of engineers in the industry, ranging from the extremely advanced to extremely basic, I would estimate that the figure is closer to 25 percent. Hence, it is a myth that ABV is a mainstream process. Increased adoption will only occur as organizations begin to invest in maturing their process capabilities.

In the early 1990's, the design community moved design up a level of abstraction from gate level to RT level. You will see evidence of this shift in Fig. 1, with the increase in the curve representing our ability to design larger blocks [3]. Yet even with today's synthesis breakthroughs in design productivity, designing and synthesizing RTL entirely from scratch cannot keep pace with what we are capable of fabricating. Hence, third-party IP that moves design to the transaction level will be necessary to increase design productivity.

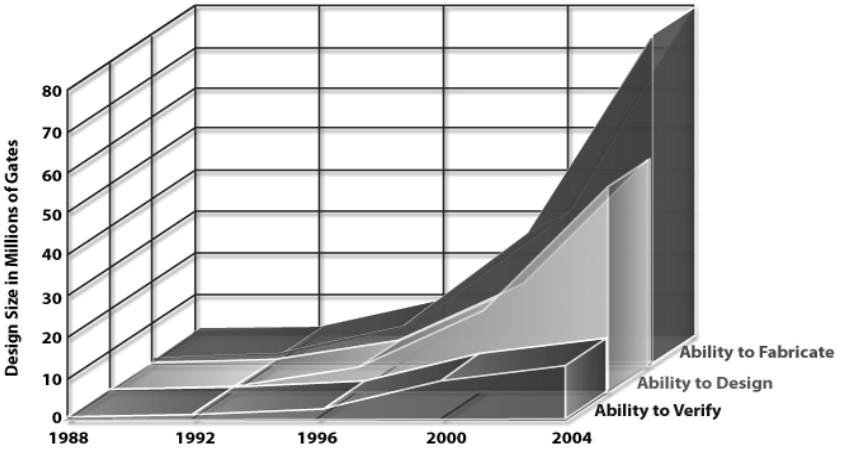

Fig. 1. Productivity gap, as reported by the Collett 2004 industry study [2]

Upon further examining Fig. 1, you might be drawn to the disparity between what we can design and what we are able to verify. Yet in many respects, the data in Fig. 1 seems to defy reality. Design teams actually do verify complex chips today, which is obvious from the myriad new electronic products available. In fact, today's verification gap is not due to a lack of innovation in verification technology. What differentiates a successful team from an unsuccessful team is process and adoption of new verification methods. Unsuccessful teams tend to approach development in an ad hoc

fashion, while successful teams employ a more mature level of methodology that is systematic.

In this tutorial, I present multiple case studies illustrating successful integration of ABV by more advanced verification teams. In addition, I present case studies that illustrate multiple challenges faced by mainstream verification teams when attempting to adopt assertion-based techniques.

4 Future Direction and Research Opportunities

The industry is currently facing a design and verification productivity crisis, as illustrated by Fig. 1. Today's RTL-based flows cannot accommodate rapid iterations in design explorations, nor can they accommodate late stage changes in design features required by the growing consumer and wireless electronics market. Historically, increases in productivity have been achieved by raising the level of design and verification abstraction. Today, industry is just beginning to witness a shift in abstraction level from RTL to transaction level. While the increase in abstraction offers many advantages, there are a number of unanswered questions in terms of how to describe design intent (that is, assertions) on transaction-level models. These unanswered questions present opportunities for future research.

In this tutorial, I present a number of ABV research opportunities, which are based on discussions with multiple tool developers and industry experts currently applying assertion-based techniques.

References

1. Abarbanel, Y., Beer, I., Gluhovsky, L., Keidar, S., Wolfsthal, Y.: FoCs—Automatic Generation of Simulation Checkers from Formal Specifications. In: Proc. 12th International Conference Computer Aided Verification, pp. 414–427 (2000)
2. 2004 IC/ASIC Functional Verification Study, Industry report from Collett International Research, p. 34 (2004)
3. EDA Market Statistics Service Report, Far West Research (2008)
4. Turing, A.: In Report of a conference on high speed automatic calculating machines, pp. 67–69, Univ. Math. Laboratory, Cambridge (1949)
5. Pnueli, A.: The temporal logic of programs. In: Proc. 18th IEEE Symp. on Foundation of Computer Science, pp. 46–57 (1977)
6. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
7. Wolper, P.: Temporal logic can be more expressive. Information and Control 56(1/2), 72–99 (1983)
8. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
9. Vardi, M.Y.: Branching vs. linear time: Final showdown. In: Margaria, T., Yi, W. (eds.) ETAPS 2001 and TACAS 2001. LNCS, vol. 2031, Springer, Heidelberg (2001)
10. Iwashita, H., Nakata, T.: Forward Model Checking Techniques Oriented to Buggy Designs. In: International Conference on Computer Aided Design, ICCAD (1997)
11. Beer, I., Ben-David, S., Landver, A.: On-the-fly model checking of RCTL formulas. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 184–194. Springer, Heidelberg (1998)

12. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The ForSpec Temporal Logic: A New Temporal Property-Specification Language. In: Katoen, J.-P., Stevens, P. (eds.) ETAPS 2002 and TACAS 2002. LNCS, vol. 2280, pp. 296–311. Springer, Heidelberg (2002)
13. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer, Heidelberg (2006)
14. Long, J., Seawright, A.: Synthesizing SVA Local Variables for Formal Verification. In: Proceedings of the 44th Design Automation Conference, DAC 2007, pp. 75–80 (2007)
15. Verification Census, extracted from the world-wide-web on April 16 (2008), http://www.deepchip.com/posts/dvcon07.html