FUNCTIONAL VERIFICATION: APPROACHES AND CHALLENGES

A. MOLINA† and O. CADENAS‡

† *Computer Architecture Department, Universitat Politècnica de Catalunya, Barcelona, Spain*
amolina@ac.upc.edu
‡ *School of System Engineering, University of Reading, Reading RG6 6AY, UK*
o.cadenas@reading.ac.uk

***Abstract*— It's a fact that functional verification (FV) is paramount within the hardware's design cycle. With so many new techniques available today to help with FV, which techniques should we really use? The answer is not straightforward and is often confusing and costly. The tools and techniques to be used in a project have to be decided upon early in the design cycle to get the best value for these new verification methods. This paper gives a quick survey in the form of an overview on FV, establishes the difference between verification and validation, describes the bottlenecks that appear in the verification process, examines the challenges in FV and exposes the current FV technologies and trends.**

***Keywords*— Functional verification, Simulation-based verification.**

I. INTRODUCTION

Functional verification (FV) is a necessary step in the development of today's complex digital designs. Hardware complexity growth continues to follow Moore's Law (Moore, 1965), but verification comple-xity is even more challenging. In fact, it theoretically rises exponentially with hardware complexity doubling exponentially with time (Dempster and Stuart, 2001). FV is widely acknowledged as a major bottleneck in design methodology: up to 70% of the design development time and resources are spent on FV (Fine and Ziv, 2003). Recent study highlights the challenges of FV (Mishra and Dutt, 2005): Figure 1 shows the statistics of the SOC designs in terms of design complexity (*logic gates*), design time (*engineer years*), and verification complexity (*simulation vectors*) (Spirakis, 2004). Recently all major EDA companies in the electronic sector are aggressively targeting the verification process with new and better EDA tools and a substantial number of seminars and workshops. This paper offers a quick review on what is being agreed is this rapid evolving area by examining recent references generated both by industry and the academia. The paper is organized as follows. Section 2 establishes the difference between verification and validation. Section 3 describes the bottlenecks that appear in the verification process. Section 4 examines the challenges in FV followed by current FV technologies and trends in Section 5. Finally, Section 6 gives some remarks as conclusions.

II. VERIFICATION VERSUS VALIDATION

Kropf (1997) defines "validation" as the "process of gaining confidence in the specification by examining the behavior of the implementation." Recently there was discussion on the subject of "verification versus validation". Many views were presented regarding the difference. One view was that "validation ensures it's the right design; while verification ensures that the design is right" (Verification Guild Website, 2006). Another view was "verification means pre-silicon testing (Verilog/VHDL simulations) while validation is post-silicon testing (testing silicon on boards in the laboratory)".

Whether it is validation or verification, two things need to happen to ensure that the silicon meets the specification: (1) The chip specification is interpreted correctly (typically through documentation and sometimes mo-deling). (2) The interpretation is captured and implemented correctly (typically through HDL) and synthesized into silicon and packaged as a chip. For the purposes of this article we will consider the second step as verification, and the first step as validation.

III. BOTTLENECKS

A. Design bottleneck

Design time is a function of silicon complexity. This gives rise to system complexity, which affects time to market, as show in Fig. 2.

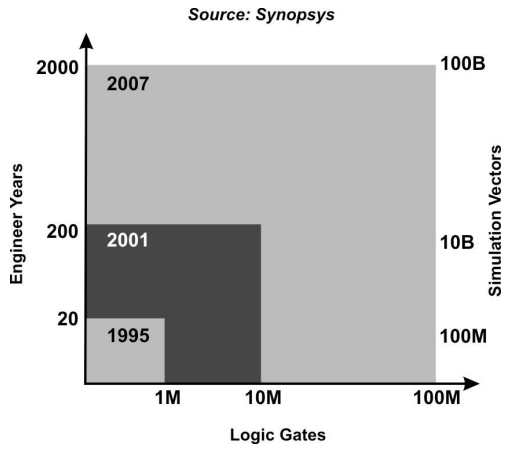

Fig. 1. The study highlights the tremendous comple-xity faced by simulation-based validation of complex SOCs: it estimates that by 2007, a complex SOC will need 2000 engineer years to write 25 million lines of register-transfer level (RTL) code and one trillion simulation vectors for functional verification.

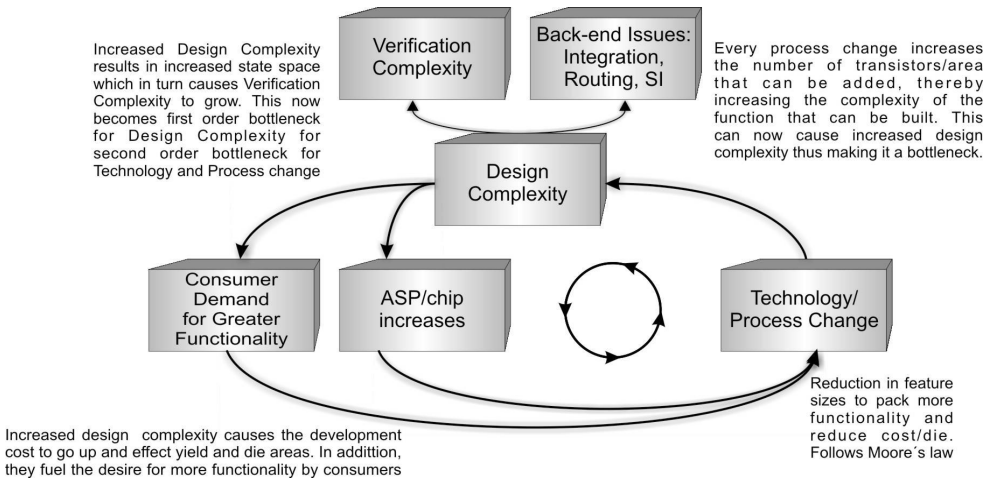

Figure 2. Technology cycle. Historically, "product time" (the time it takes for a concept to become a production part) has been mainly a function of design time. Whenever new technology or process is introduced, design time was and continues to be the primary bottleneck.

Following an exponential increase in the number of transistors in designs, a linear increase in compute time or number of engineers was not adequate to reduce design time. To solve this problem, the electronic design automation (EDA) industry stepped in to introduce the concept of design abstraction through automation. Language-based solutions such as Verilog and VHDL were introduced. The IEEE has defined standards for both (IEEE Standards, 1076-2002 and 1364-2001), and all major EDA vendors support both languages equally well (Taylor *et al.,* 1998; Acellera Website, 2006). The latest languages being widely accepted and supported by EDA world are SystemC and SystemVerilog. For the current technology processes, design complexity is well understood. Design bottleneck has been overcome to some extent thanks to the productivity gains through the use of EDA tools. Having solved the first round of problems, the focus now is on solving the effects of the first order problems such as the verification bottleneck.

B. Verification bottleneck

The verification bottleneck is an effect of raising the design abstraction level for the following reasons (See Figure 3). Designing at a higher abstraction level allows us to build highly complex functions with ease. This increase in design complexity results in almost doubling the verification effort. Functional complexity has been doubled and hence its verification scope.

Using a higher level of abstraction for design, transformation, and eventual mapping to the end product is not performed without information loss and misinterpretation. For instance, synthesis takes an HDL-level design and transforms it to the gate level. Verification is needed at this level to ensure that the transformation was indeed correct, and that design intent was not lost. Raising the level of abstraction also brings about the question of interpretation of the code that is used to describe the design during simulation. Other factors that affect the verification problem are:

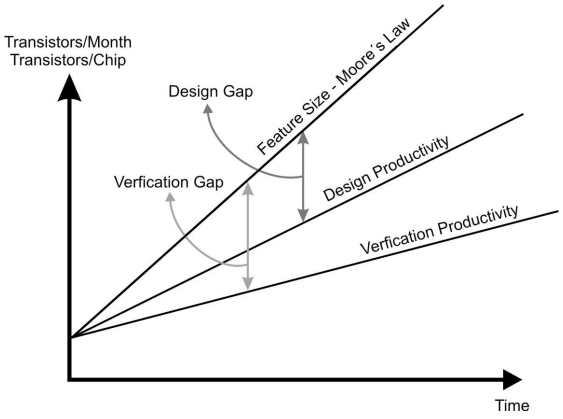

Figure 3. Design and Verification Gaps. Design productivity growth continues to remain lower than complexity growth — but this time around, it is verification time, not design time, that poses the challenge. A recent statistic showed that 60-70% of the entire product cycle for a complex logic chip is dedicated to verification tasks (Warren, 2002).

Increase in functional complexity because of the heterogeneous nature of designs today; for example, co-existence of hardware and software, analog and digital. The requirement for higher system reliability forces verification tasks to ensure that a chip level function will perform satisfactorily in a system environment, especially when a chip level defect has a multiplicative effect. To increase verification productivity, the EDA industry came up with a solution similar to what was used to solve the design bottleneck — the concept of abstraction. High-level language constructs were embedded into Verilog and VHDL to help in verification; these included constructs such as tasks, threading (fork, join) and control structures. This provided more control to fully exercise the design on all functional corners. However, these constructs were not synthesizable and hence not used by designers as part of actual design code. As complexity continued to grow, new verification languages were created and introduced that could

verify complex designs at various levels of abstraction. Along with new verification languages came technologies and tools that supported them.

IV. THE VERIFICATION CHALLENGES

The verification engineer faces four major challenges: dealing with enormous state space size, detecting incorrect behavior, lack of a golden reference model and lack of a comprehensive functional coverage metric. The scale of the state space is the first verification challenge. To verify exhaustively that a chip is functionally correct, the verification engineer needs to check that each possible current state and each possible input combination yields the correct next state. To combat state space explosion, verification engineers break the problem down into smaller pieces. Rather than verifying the entire chip at once, the verification team will tackle subcomponents of the design and verify these pieces separately. Once the smaller, more manageable pieces are verified, the team stitches the chip subcomponents back together and ensures that they work. The second verification challenge is detecting when the design violates the expected behavior or specification. With all of the possible transitions from one state to the next, the verification engineer must be able to identify whether or not the design acted correctly based on the current state and input. Rather than focusing on each of the possible states of the hardware, verification engineers validate the logic at a higher level of abstraction: inputs are grouped into valid command and data sets, and the verification engineer concentrates on the behavior of the design based on the functional input stimulus.

Figure 4 summarizes a study of the pre-silicon logic bugs found in the Intel IA32 family of microarchitectures. This trend again shows an exponential increase in the number of logic bugs: a growth rate of 300-400% from one generation to the next. The bug rate is linearly proportional to the number of lines of structural RTL code in each design, indicating a roughly constant density (Fine and Ziv, 2003).

The next obvious question is – where do all these bugs come from? An Intel report summarized the results of a statistical study of the 7855 bugs found in the Pentium 4 processor design prior to initial tape out (Fine and Ziv, 2003) (see Fig. 5).

Typically, there are two fundamental reasons for so many logic bugs: lack of a golden reference model and lack for a comprehensive functional coverage metric. First, there are multiple specification models above the RTL level (functional model, timing model, verification model, etc.). The consistency of these models is a major concern due to lack of a golden reference model. Second, the design verification problem is further aggravated due to lack of a functional coverage metric that can be used to determine the coverage of the microarchitectural features, as well as the quality of functional validation. Several coverage measures do not have any direct relationship to the functionality of the design. For example in the case of a pipelined processor, none of these measures determine if all possible interactions of hazards, stalls and exceptions are verified. In simplest terms, then, the verification challenge comes down to two fundamentals: (1) Drive the state transitions and input scenarios. (2) Flag any incorrect behavior exhibited by the design.

V. CURRENT VERIFICATION TECHNOLOGIES

Figure 6 shows a snapshot of the various methods and technologies that are available to companies today.

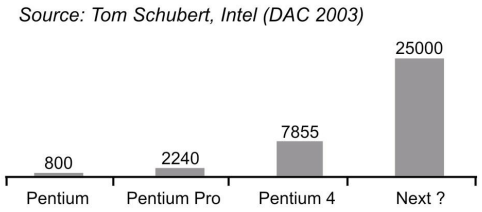

Figure 4. Pre-silicon logic bugs per generation. Simple extrapolation indicates that unless a radically new approach is employed, we can expect to see 20-30K bugs designed into the next generation and 100K in the subsequent generation. Clearly – in the face of shrinking time-to-markets – the amount of validation effort rapid-ly becomes intractable, and will significantly impact product schedules, with the additional risk of shipping products with undetected bugs.

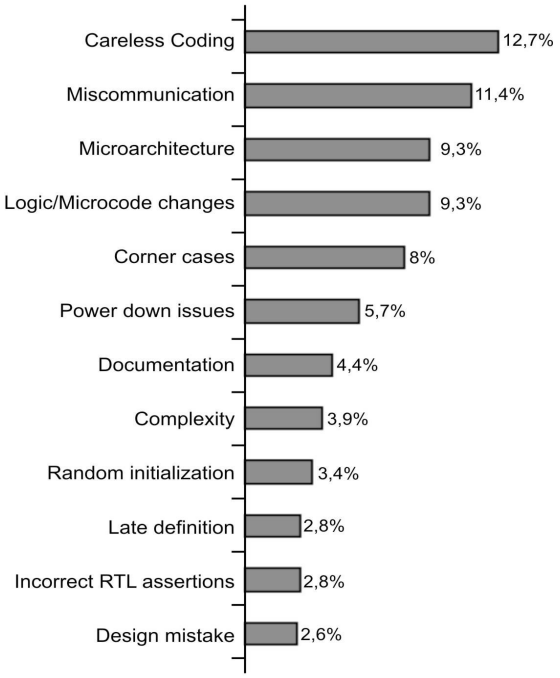

Figure 5. Although "complexity" is ranked eighth on the list of bug causes, it is clear that it contributes to many of the categories listed above. More complex architectures need more extensive documentation to describe them; they require larger design teams to implement them, increasing the likelihood of miscommunication between team members; and they introduce more corner cases, resulting in undiscovered bugs. Hence, microarchitectural complexity is the major contributor of the logic bugs.

When the logic gets more complex, the verification space increases. This brings about random dynamic simulation, which provides random stimulus to the design in an effort to maximize the functional space that can be covered. The problem with random testing is that for very large and complex designs, it can be an unbounded problem. To solve this problem, the EDA industry introduced higher-level verification languages such as Open Vera (Open Vera Website, 2006; Hollander *et al.,* 2001), and SVL (SystemC Verification Library) (SystemC Website, 2006). These introduced concepts such as constrained-random stimulus, random stimulus distribution and reactive test benches. In addition to the introduction of randomization features, new verification languages and tools increased productivity by decreasing the amount of time engineers spent on building various test case scenarios for stimulus generation. For example, the test scenarios can be written at the highest level of abstraction and can be extended to any lower level of abstraction by using powerful object-oriented constructs.

Designers use assertions as placeholders to describe assumptions and behavior (including temporal) associated with a design. Assertions get triggered during a dynamic simulation if the design meets or fails the specification or assumption. Assertions can also be used in a formal/static functional verification environment.

In order to make sure that the gate level representation is the same as the HDL implementation, an "equivalence check" is performed by using matching points and comparing the logic between these points. A data structure is generated and compared for output value patterns for the same input pattern. If they are different, then the representations (in this case gate and RTL) are not equivalent. Equivalence checking is sometimes performed between two netlists (gate level) or two RTL implementations when one of the representations has gone through some type of transformation.

VI. CONCLUSIONS

Functional design verification is an art, an art of combining hardware and software and communication skills with creative strategies to understand a design and its usage context to ensure that the design's quality and delivery schedule are successful. Communication and monitoring are critical not only in understanding the design and where its bugs lurk, but also are optimally balancing how to apply one's energies to cover testing of the design thoroughly.

Many of the problems associated with the functional verification methodologies of today are based on the absence of an effective automation to combat the discouraging growth in the size and complexity of the design. This has forced to rely on manual effort in the development of environments for tests. The examination and the treatment of the results of the test is also a manual procedure. Perhaps the most notorious problem facing the engineers of design and verification is the lack of effective metric to measure the progress of the verification. Code coverage for example, indicate lines of verification code that was visited in a simulation, but it does not offer any indication of which functionality it was verified. As result, the engineer never is sure if a sufficient quantity of verification has been realized.

The biggest of all the efforts in the verification is to determine a comprehensive methodology of the verification capable of verifying arbitrary designs. For now, the majority of the efforts are contained in developing and improving specific skills, each of which is excellent in some area of the verification. A common theme of verification effort is to find a methodology comprehensive of verification.

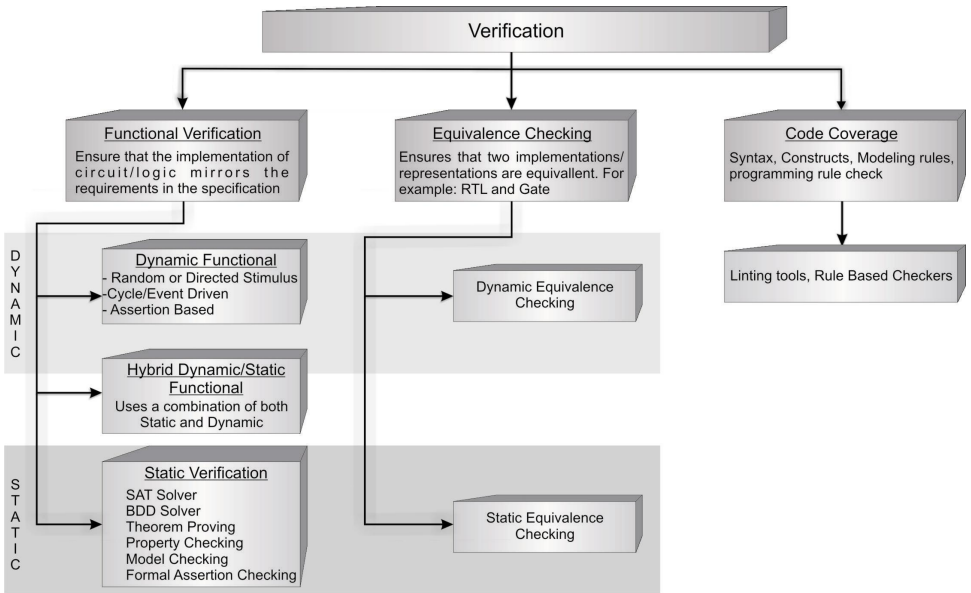

Figure 6. Verification Methodologies. The most widespread method of functional verification is dynamic in nature. The reason it is called "dynamic" is because input patterns/stimulus are generated and applied over a number of clock cycles to the design, and the corresponding result is collected and compared against a reference/golden model for conformance with the specification.

REFERENCES

Accelera Website, *http://www.acelera.org,* (2006).

Dempster, D.J. and M.G. Stuart, *Verification Methodology Manual: Techniques for Verifying HDL Designs*, second edition. Teamwork Int. (2001).

Fine, S. and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," *Proc. 40th Design Automation Conf. (DAC)*, Anaheim, CA, USA, 286-291 (2003).

Hollander, Y., M. Morley and A. Noy, "The e Language: a fresh separation of concerns," *Proc. 38th Technology of Objetc-Oriented Languages and Systems Conf. (TOOLS)*, Zurich, Switzerland, 41-50 (2001).

IEEE Standards, *VHDL Language Reference Manual* (IEEE Std. 1076-2002). *Verilog Hardware Description Language* (IEEE Std. 1364-2001).

Kropf, T., *Formal Hardware Verification: Methods and Systems in Comparison*, Springer-Verlag, London (1997).

Mishra, P. and N. D. Dutt, *Functional Verification of Programmable Embedded Architectures. A top-Down Approach,* Springer, USA (2005).

Moore, G., "Cramming More Components onto Integrated Circuits," *Electronics Magazine*, **38**, 114-117 (1965).

Open Vera Website, *http://www.openvera.org*, (2006).

Spirakis, G.S., "Opportunities and Challenges in Building Silicon Products In 65nm and Beyond," *Proc. Design Automation and Test in Europe Conf. and Exhibition (DATE'04)*, Paris, France, 2-3 (2004).

SystemC Website, *http://www.systemc.org,* (2006).

Taylor, S., M. Quinn., D. Brown., N. Dohm., S. Hildebrandt., J. Huggins and C. Ramey, "Functional verification of a multiple-issue, out-order, superscalar Alpha processor: The DEC Alpha 21264 microprocessor," *Proc. 35th Design Automation Conf. (DAC)*, San Francisco, California, USA, 638-643 (1998).

Verification Guild Website, *http://www.verificationguild.com,* (2006).

Warren, A.H., "Introduction: Special Issue on Microprocessor Verifications," *J. Formal Methods in System Design. USA,* **20**, 135-137 (2002).

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.