Analysis and Discussion of Simulation Verification Techniques for Chip Design: Postprint

Authors: Lü Tao, Li Huawei, Li Xiaowei

Date: 2017-03-09T00:00:00+00:00

Abstract

This paper analyzes the dilemmas encountered by simulation verification technology with the development of design scale and their underlying causes, arguing that design verification based on error models represents a promising technical direction for breakthrough progress. The paper then comprehensively surveys the current research status of error model-based design verification technology from three perspectives, along with some of our work in this domain. Firstly, it elaborates on the research status of design error models, focusing on the concept of differential testing and the omission error model along with its testing methodologies. Secondly, it introduces the error injection system ErrorInjector, which supports a rich variety of design error models and features well-designed extensible interfaces, serving as a highly beneficial foundational platform for design error model research. Finally, it presents a static observability quantification analysis method based on error masking probability, and a methodology for selecting internal observation points according to low observability root causes, thereby providing an illustrative example for investigating design-for-verifiability technology from the perspective of effects induced by design errors.

Full Text

Vol. 8 No. 2 Information Technology Express Vol.8 No.2 Analysis and Discussion of Simulation Verification Technology for Chip Design[1] Tao Lv, Huawei Li, Xiaowei Li

Abstract

This paper analyzes the dilemmas faced by simulation verification technology as design scales continue to grow and their underlying causes, arguing that design verification based on error models represents a promising direction for breakthrough progress. The paper then comprehensively introduces the current research status of error model-based design verification technology from three

perspectives, along with some of our work in this area. First, it elaborates on the research status of design error models, focusing on the concept of mutation testing and the item-missing error model and its testing methodology. Second, it introduces the ErrorInjector error injection system, which supports a rich variety of design error models and features well-designed extensible interfaces, serving as a highly beneficial foundational platform for design error model research. Finally, it presents a static observability quantification analysis method based on error-masking probability and, building upon this, a method for selecting internal observation points according to low-observability root causes, providing an example for studying design-for-verification techniques from the perspective of effects triggered by design errors.

Keywords: chip design, simulation verification, error model, error injection, verifiability

With the continuous development of integrated circuit technology, chip design scales are becoming increasingly larger and more complex, imposing higher requirements on functional correctness, performance, power consumption, and reliability. Among these, functional correctness is the most fundamental requirement for chip design and must be addressed through design verification technology, making verification a critical supporting technology for chip design.

In the early stages of integrated circuit development, design verification was typically accomplished through customized or even manual approaches—for example, designers hand-wrote test cases or ran typical application programs to verify chip design correctness. As design scales continued to expand, design verification research has attracted increasing attention from academia, particularly over the past decade, becoming a hot research topic. This is primarily because the processing capabilities of existing verification technologies can no longer meet the demands of growing design scales. According to a report by Collett Market Research, functional errors became the leading cause of chip respins in 2002, and this factor' s proportion continued to grow in 2004.

The most direct factor affecting design verification technology's processing capability is the functional complexity of chip design. However, this factor is difficult to quantify through any single metric. Common metrics such as transistor count or logic gate count cannot comprehensively reflect a chip design' s functional complexity. The 2004 International Technology Roadmap for Semiconductors (ITRS) states: "Design scale grows exponentially with Moore' s Law. If we measure functional complexity by the number of different states in the system that require verification, then in the worst case, functional complexity also grows exponentially with design scale. This leads to a double-exponential explosion [1]." Evidently, the problem scale growth rate in design verification far exceeds that of design scale itself.

In addition to verifying design states, design verification technology often requires reasoning over state sequences, where the problem scale expands even more rapidly. According to ITRS 2008 predictions, the number of transistors

integrated on a chip will reach tens of billions by 2019. Increasing design scales will pose tremendous challenges to design verification technology. Consequently, ITRS 2008 notes that without major breakthroughs in verification technology, the semiconductor industry' s future development could stagnate [1].

Existing design verification technologies can generally be divided into two major categories from a methodological perspective: simulation-based verification and formal verification. Simply put, simulation-based verification discovers design errors by applying input vectors and analyzing chip design behavior in a simulator, while formal verification uses mathematical methods to rigorously prove that a chip design meets given requirements (i.e., specifications). The combination of these two approaches has also given rise to semi-formal verification techniques.

Simulation verification has been the dominant verification technology consistently adopted by industry. Formal verification can implicitly exhaust the entire circuit' s input variable space to prove functional correctness. Although its importance is growing, formal verification technology is more sensitive to design scale and has not yet become the primary method for industrial design verification. This paper will analyze the challenges currently faced by simulation verification technology and subsequently introduce error model-based design verification technology—a promising direction for breakthrough progress.

1 Challenges in Simulation Verification

This section first briefly introduces simulation verification technology, then analyzes the challenges it faces, and expounds on the significance of researching error model-based design verification.

1.1 Introduction to Simulation Verification

Simulation verification has become the most commonly used and dominant verification technology in industry primarily due to its powerful scalability. Generally speaking, designs of any scale can always be verified through simulation. The languages used to develop simulation verification environments (testbenches) have undergone several generations of evolution—from directly using hardware description languages (HDL), to object-oriented languages (such as C++), to specialized verification languages (such as SystemVerilog and e). Specialized verification languages improve verification efficiency, enabling verification engineers to develop powerful verification environments quickly and conveniently. A typical verification environment in simulation is shown in the dashed portion of Figure 1 [Figure 1: see original paper]. The modules within the dashed line constitute the verification environment components, nodes outside the dashed line represent documents related to the simulation process, and the connecting lines indicate information or data flow relationships.

In the simulation verification process, verification engineers must first understand the functional specifications and develop corresponding verification plans,

then create test cases according to these plans. In addition to customizing specific test cases, commonly employed techniques include constraint-based random vector generation. To improve development efficiency and module reusability in verification environments, verification languages typically adopt an object-oriented language-like approach to describe test cases. Therefore, when applying test cases to the design under verification (DUV), especially at the register-transfer level (RTL) or gate level, it is necessary to map relatively abstract test cases into binary vectors[2], and when necessary, apply these binary vectors through a bus functional model (BFM) [2] according to the timing relationships required by the DUV—this is the function implemented by the input control portion in Figure 1. The output collection portion performs the inverse function of input control. For data collected during simulation, on one hand, result checking (including timing and data checks) is required to form result check reports reflecting whether design errors were discovered during simulation; on the other hand, coverage analysis is needed to form coverage reports assessing the adequacy of simulation verification. Detailed introductions to various components in design verification environments can be found in [3].

1.2 Challenges in the Nanometer Era

As previously explained, the problem scale in design verification grows double-exponentially with Moore' s Law. When integrated circuit process dimensions develop to the nanometer scale and chip design scales reach tens of billions of transistors, the problem scale in design verification will become unmanageable for existing technologies.

According to the introduction in Section 1.1, the main components of simulation verification include: (1) stimulus generation, (2) simulation, (3) result checking, and (4) coverage evaluation. Among these, result checking is typically implemented through assertions or comparison with reference models, and the related technologies are relatively mature. In the other three components, increasing simulation speed helps verify larger functional spaces in less time. For example, hardware acceleration methods can improve simulation speed by several orders of magnitude, but this still cannot keep pace with the growth of functional complexity. Therefore, research on simulation verification focuses on stimulus generation and coverage evaluation.

Functional coverage evaluation is the most commonly used and important method in industry for measuring verification progress. However, defining high-quality functional coverage points requires engineers to possess substantial expert knowledge and relevant experience, representing the primary shortcoming of functional coverage evaluation. In other words, the effectiveness of functional coverage metrics depends on the completeness of the defined functional coverage points, and currently, no method exists to verify this completeness. Applying coverage evaluation methods during simulation verification can identify coverage holes, providing important guidance for stimulus generation. If functional coverage point definitions are incomplete, it becomes

very difficult to compensate for this deficiency through stimulus generation effectiveness without adequate coverage information guidance.

Stimulus generation in simulation verification is typically performed at the register-transfer level and above. In this context, stimulus generation itself is extremely difficult for three main reasons: (1) properties to be verified often involve multiple clock cycles, resulting in a vast problem space; (2) correspondingly, stimulus generation in design verification often targets sequential circuits, unlike circuit testing where combinational logic is primarily handled (through testability design techniques such as scan chains); and (3) designs at the RTL level and above often contain bit-vectors or word-level variables with much larger value ranges than Boolean quantities at the gate level, making corresponding operations more complex. These three factors lead stimulus generation in the design verification domain to typically avoid deterministic algorithms, instead mostly adopting random algorithms [4] or simulation-based methods [5][6]. If functional coverage point definitions are incomplete, it becomes very difficult to discover design errors in related logic within a short time using random or simulation-based stimulus generation methods. For example, Intel claimed that the floating-point division error in Pentium chips would only be encountered once every 27,000 years by a typical spreadsheet user [7].

In summary, stimulus generation for simulation verification is extremely difficult and requires guidance from coverage information. However, functional coverage demands excessive manual involvement, and its target space is too large, growing exponentially with design scale. Consequently, existing simulation verification technologies centered on functional coverage can hardly meet the future development needs of chip design.

1.3 Significance of Error Model Research

Discovering design errors is the most direct objective of simulation verification. In the integrated circuit testing domain, the successful proposal of the stuck-at fault model has enabled high automation in test generation, testability analysis, and fault coverage evaluation, with test completeness being well-founded. Similarly, if high-quality design error models can be proposed, they are expected to greatly advance related research in simulation verification. This viewpoint is also reflected in ITRS 2007 [1].

First, design error model research is expected to reduce the problem scale of design verification technology. As previously mentioned, design verification from a functional perspective exhibits exponential growth in problem scale with design scale. However, if design verification is conducted from an error perspective, since many error models are built by analyzing code changes caused by certain types of design errors, the problem scale essentially becomes of the same order of magnitude as the design scale (code).

Second, design error model research can form a tight complement with existing functional coverage-centered simulation verification technologies. On one

hand, error model-based design verification cannot be separated from basic functional evaluation, as the ultimate goal of design verification is to ensure the design meets its functional specifications. On the other hand, existing functional coverage-centered simulation verification technologies struggle to achieve complete coverage of various boundary cases due to excessively large problem scales, while error model-based technologies can compensate for this from the perspective of potential errors in the code, creating a situation where the two approaches complement and promote each other. Although existing coverage evaluation methods such as statement coverage [6] and branch coverage [8] also assess verification completeness from a code perspective, these methods typically employ heuristic ideas and lack direct connection to design verification effectiveness. Consequently, their evaluation effectiveness has always lacked sufficient proof, with 100% code coverage typically serving only as a basic requirement for simulation verification. Error model-based design verification technology holds inherent advantages in this regard. In other words, discovering design errors is the most direct way to demonstrate verification effectiveness. When 100% coverage of a certain error model is achieved, we can usually state that the design contains no errors of that class.

In short, error model-based design verification technology is an effective method for mainstream simulation verification technologies to address challenges brought by design scale development and has received widespread attention from industry.

2 Design Error Models

In the chip design flow, various reasons can lead to functional errors, such as errors caused by poor version management, mismatches between implementation and specifications due to functional specification changes, and connectivity issues caused by interface incompatibility. This paper does not intend to explore all possible design errors but is limited to functional errors introduced by designers.

2.1 Related Research on Error Models

As shown in Table 1 , from a modeling perspective, research on error models can be divided into three categories.

The first category is explicit error models, which describe design errors through changes they cause in design code. Mutation testing is a typical method in this category of error model research. First proposed by software testing scholars in [9], mutation testing' s main idea is to make minor changes to the original design to generate various mutants, then generate test vectors that can distinguish the original design from its mutants. If the original design behaves correctly under the test vectors, it does not contain the design errors corresponding to the respective mutants.

Error models in mutation testing are extracted from common design errors, such

as mistakenly writing a logical "AND" operation (&) as a logical "OR" operation (|), or writing "greater than or equal to" (>=) as "greater than" (>) in relational operations. Each mutant contains a single modeled error, which facilitates error coverage calculation and is based on two assumptions—the competent programmer hypothesis and the coupling effect hypothesis. The "competent programmer hypothesis" suggests that a competent programmer' s code is always close to the correct design, and if errors occur, they are relatively simple ones. The "coupling effect hypothesis" posits that stimulus sets generated for simple errors can also discover complex errors composed of simple errors. This hypothesis was proposed in [9] in 1978, and software testing scholars have investigated it experimentally [10] and through theoretical analysis [11].

Reference [12] reports on Mothra, an automated mutation testing system for Fortran programs. Due to the excessive complexity of software programs and the huge number of mutants (for example, Mothra generated 970 mutants for a 27-line program [12]), mutation testing incurred excessive overhead and thus was not widely applied in software testing. Nevertheless, mutation testing laid the foundation for research on design error models, and its ideas have been borrowed for hardware design verification technologies [13].

Barclay et al. proposed control fault models for if-then-else, case-when, and signal assignment structures in VHDL[3] code. Predicates in these conditional structures are required to be stuck-at-1 and stuck-at-0 during verification, while assignment operations are required to be open to detect dead clauses [14]. Reference [15] defines four simple error models for gate-level combinational circuits: gate type errors, gate number errors, input number errors, and input signal errors, and maps error models to single-stuck line faults (SSL) to leverage Automatic Test Pattern Generation (ATPG) tools for stimulus generation. Reference [16] proposes higher-level design error models, including bus order errors and bus source errors, and manually generates test sets for register-transfer level circuit designs.

The second category is abstract error models. The main idea is to abstract a certain model of circuit design and analyze and extract functional error models on this basis. For example, Min Yinghua et al. proposed a register-transfer level model and, based on this, proposed multiple error models, including decoding faults, data storage and transmission faults, etc. [17]. Shen Li et al. also proposed corresponding functional error models for microprocessor design [18].

The third category is implicit error models, which analyze design error characteristics from the perspective of effects triggered by errors. For example, [19] expresses circuit design schemes in polynomial form and analyzes design errors using polynomial zero-point theory. Reference [20] proposes a probability calculation method based on masked value sets (MVS) to evaluate observability-enhanced statement coverage. Our proposed static observability analysis method also assesses the impact of design errors from the perspective of variable value changes they cause [21].

The above research on design error models mainly concentrated in the 1980s and 1990s, with few major advances in the past decade. Recent research primarily focuses on modeling specific design errors—for example, scholars from Peking University proposed a cross-clock domain error model for the effect of metastability in multi-clock domain SoC designs [22][20]. As design scales continue to expand, circuit design development urgently demands verification technologies. Against this backdrop, error model-based design verification is expected to form a tight complement with existing mainstream simulation verification technologies, thus holding promising development prospects.

2.2 Item-Missing Error Model

In our practical chip design verification processes, we discovered design errors involving missing clauses in expressions. Such design errors also exist in other chip designs. Scholars from UCLA analyzed Intel' s Pentium II microprocessor errata and found that among the 73 functional defects reported for Pentium II microprocessors between May 1997 and April 1999, most were control errors. These control errors can be roughly divided into two categories: missing control functionality or incorrect control functionality [23]. We found that design errors involving missing expression clauses align with the basic understanding in [23] and represent a new expansion—besides control logic, missing clauses may also occur in the right-hand side expressions of assignment statements. However, existing error models and testing methods cannot handle such situations. In [24], we proposed a new design error model—the item-missing error (IME) model—based on analysis of design errors in actual engineering projects, which can handle such missing clause errors in expressions. In this paper, we only use descriptive language to illustrate the IME model and its testing methodology; more detailed and formal elaboration can be found in [24].

The proposed IME model can model errors in hardware design caused by missing clauses, including missing clauses in right-hand side expressions of continuous assignments, non-blocking assignments, and blocking assignments, as well as missing clauses in conditional expressions of if statements, case statements, and conditional operators (i.e., the ternary operator).

Below, we explain the practical significance of the IME model using actual design errors from projects. Figure 2 Figure 2: see original paper shows code containing a design error, while Figure 2(b) shows the corrected code. This code snippet is from the reset logic of an AMBA AHB master interface. Besides the initialization operations of the master interface logic itself, there is another scenario requiring reset of the master interface logic—when a slave returns an ERROR response, the master interface logic also needs to be reset. This error can be modeled using the IME model.

According to the AMBA AHB protocol specification [25], an ERROR response requires at least two clock cycles to represent: in the second-to-last clock cycle, the slave sets the HRESP[1:0] signals to 2' b01 and simultaneously pulls

HREADY low; in the last clock cycle, the slave keeps HRESP[1:0] unchanged while pulling HREADY high. Evidently, recognizing an ERROR response requires relatively complex logic, so fixing the missing clause error may require operations on some signals rather than simply adding an internal signal or input signal (the bus_error signal added in Figure 2(b) is generated to recognize the two-cycle pattern corresponding to the ERROR response). Such design errors cannot be discovered through mutation testing techniques. In other words, mutation testing mutates existing code and cannot handle clauses that are missing and therefore do not appear in the code. Consequently, item-missing errors require new testing methods for detection.

We propose a constraint-based random stimulus generation method to detect item-missing errors. Using the code in Figure 3 [Figure 3: see original paper] as an example, the expression "a&b" in the if statement on line 1 of Figure 3(a) contains an item-missing error; its correct form is "a&b&c" as shown in Figure 3(b). When a=1, b=1, and c=0, the erroneous code in Figure 3(a) will execute the if branch on line 2, while the correct code in Figure 3(b) will execute the else branch on line 4. If the value of "input1|input2" does not equal "input1|input2" at this time, this item-missing error can be discovered. The insight from this example is that if existing terms are constrained to take their non-controlling values during stimulus generation (the definition of non-controlling values can be found in [24]), the effect of the missing term may be exposed.

In actual design verification processes, the missing term cannot be known in advance. For example, in the case shown in Figure 3(a), the correct code might be "a&b&c" , "a&b&d" , or "a&b&(!e)" . Therefore, the goal of our proposed IME model testing method is not to find the correct code but to increase the probability of discovering such design errors. To expose the effect of the missing term as much as possible, the effects of existing terms must be minimized as much as possible. Obviously, for a single item-missing error in an expression, stimuli that satisfy the non-controlling value constraints for existing terms have a higher probability of discovering the missing error compared to stimuli that do not satisfy these constraints, given other identical conditions. If an existing subexpression takes a controlling value, the entire expression's value will depend on that controlling value regardless of whether the missing term' s effect is triggered. Such stimuli cannot detect the item-missing error.

Based on industry-standard constraint-based random verification techniques, we propose a testing method for the IME model, with its flow shown in Figure 4 [Figure 4: see original paper]. The core steps include: first, selecting a potential item-missing error; second, obtaining non-controlling value constraints for other subexpressions relative to that error, appending these constraints to the functional constraints of the verification environment, generating random vectors that satisfy these constraints, and performing several tests on the item-missing error; third, if other potential item-missing errors exist, returning to the first step, otherwise ending the test.

As shown in Figure 5 [Figure 5: see original paper], experimental data from ac-

tual embedded processor designs demonstrates that for item-missing errors, the average detection probability with IME structural constraints added is several times higher than without IME structural constraints, reaching up to more than six times higher. This indicates that structural information extracted from the IME model is very useful for discovering such design errors and can greatly improve functional verification efficiency. Detailed experimental data and analysis can be found in [24].

3 Design Error Injection Platform ErrorInjector

Since no widely recognized design error model currently exists, and consequently no efficient design error simulator is available, the fundamental method for analyzing and comparing the advantages and disadvantages of various design error models and evaluating design error coverage is error injection and logic simulation. To our knowledge, no open-source error injection system for hardware design currently exists, causing related research teams to develop their own systems and resulting in redundant consumption of time and resources.

Therefore, we designed and implemented a design error injection platform—ErrorInjector. In error model-related research, an important characteristic of a good design error injection system is its ability to support a rich variety of error model types. It is even better if the system can support flexible error injection locations, enabling random injection throughout the entire design or concentrated injection in regions of interest. ErrorInjector possesses these features. Specifically, ErrorInjector' s main characteristics are:

1. It extracts design information through compilers and supports multiple error types. The system supports Verilog language and uses the open-source compiler Icarus Verilog [26] for design information extraction.
2. It supports both random selection and equal-spacing selection for error injection locations. The system includes a preprocessing step that counts the number of injection locations corresponding to various error models in the design and provides this information to users for describing error injection configuration.
3. It implements extensible interfaces for error models based on dynamic link library technology, giving the system good extensibility. Users only need to construct interface functions (i.e., recognition functions and injection functions related to specific error models) and compile them into dynamic link library files to integrate new error models into the ErrorInjector system. This is a user-friendly extension method that allows users to extend required error models without understanding all implementation details of ErrorInjector, greatly reducing user workload and improving system usability and extensibility.

The error injection flow of the ErrorInjector system is shown in Figure 6 [Figure 6: see original paper]. The ErrorInjector system requires two inputs: the first is the design under verification, i.e., the hardware design implemented in

Verilog language; the second is the error information specified by verification personnel, including the error model to be injected, the number of errors, and error locations. The output is multiple design files with single errors, each being syntactically correct. The current system is implemented in C++ on the Linux platform. Its user interface is shown in Figure 7 [Figure 7: see original paper]. Users can understand all possible error information in the chip design through the graphical interface and configure specific error types, numbers, and locations for injection.

Based on this platform, two types of research can be conducted:

1. **Analysis and comparison of various design error models' advantages and disadvantages.** For stimulus sets that have already discovered certain design errors in actual chip verification, error injection can be performed and simulation conducted using these stimulus sets to analyze various error models' ability to model actual design errors, thereby extracting representative error models to guide design verification technology.
2. **Evaluation of error coverage to guide stimulus generation.** Through single-error injection into chip designs, multiple syntactically correct designs with design errors can be obtained. These designs can then be simulated using logic simulators or formally verified using formal tools. Finally, the verification results are analyzed and statistically processed to calculate the ratio of discovered design errors to all injected design errors, i.e., the error coverage. Analyzing the resulting coverage holes provides important guidance for stimulus generation.

4 Design-for-Verification Techniques Based on Implicit Error Models

In our research on error models, we have deeply recognized that research on design error modeling faces many difficulties, primarily because chip design manufacturers rarely disclose details of design errors in their projects. Although chip manufacturers such as Intel provide product errata, these are typically described from a user' s perspective and are difficult for design verification research to utilize. Scholars from the University of Michigan systematically collected design error data from student microprocessor design course projects and analyzed it in [27]. Reference [28] also reports design errors discovered during formal verification of student designs. However, such publicly available data remains scarce. Moreover, errors made in student designs cannot fully represent errors in industrial designs. Considering the objective difficulties faced by explicit error modeling research, we need not confine ourselves to specific design error models but can instead conduct research from the perspective of effects triggered by design errors. This is particularly true for design-for-verification.

For design-for-verification, the approach is typically not bound to a specific explicit error model but rather analyzes verification difficulty from the perspective of effects triggered by errors. We have studied quantification analysis meth-

ods from the perspective of design error propagation and proposed a static observability quantification analysis method based on error-masking probability (EMP). Building upon this method' s quantitative analysis values for static observability of internal signals in a design, we have proposed an internal observation point selection algorithm. This algorithm can locate sources causing local design observability difficulties before simulation. Using such sources as internal observation points can enhance the ability of design verification to discover errors.

4.1 Motivation for Static Observation Point Selection Methods

Figure 8 [Figure 8: see original paper] is a schematic diagram illustrating the role of static observability analysis methods. The horizontal axis represents simulation time, and the vertical axis represents verification effectiveness. Common methods reflect verification effectiveness through coverage. There are two coverage curves in the figure. The relatively smooth upper curve shows coverage growth when internal observation points are added using static analysis methods. The lower stepwise curve shows coverage growth when internal observation points are added using traditional methods.

Traditional methods typically conduct simulation for a period first; uncovered portions during simulation are considered difficult to observe, and verification personnel add internal observation points for them before continuing simulation. This process repeats. As illustrated in Figure 8, the two solid circles on the horizontal axis indicate when internal observation points are added using traditional methods. Since points are added during simulation, new internal observation points can only take effect in subsequent simulation, causing the coverage curve to grow in steps. Moreover, for numerous uncovered regions during simulation, selecting which to use as internal observation points depends entirely on human decision-making, lacking stability and resulting in relatively slow coverage growth.

Our proposed static observability analysis method addresses this problem by not requiring simulation data for analysis. Therefore, it can report sources causing observability difficulties in certain regions before simulation begins, prompting verification personnel to add them as internal observation points. This avoids the potential blindness caused by human dependency in traditional methods, and newly added internal observation points can take effect when simulation begins, enabling smooth and rapid coverage growth.

4.2 Method for Selecting Internal Observation Points Based on Low-Observability Sources

Observability in design verification discussed in this paper refers to the difficulty of observing error effects on a signal within a design. Static observability (SOBS) is a specific quantitative metric used to evaluate the observability of an internal signal in hardware design. For a given internal signal, its SOBS value represents

the likelihood that a design error on that signal will be masked.

We estimate the difficulty of design error propagation from the perspective of operand values. Since design error effects are typically reflected as impacts on signal values in a design, and different test cases execute different paths in the design and encounter different operations, signal values propagate with varying difficulty across different register-transfer level operations. This leads to different design error propagation effects under different test cases. Calculation methods for error-masking probabilities of typical word-level operations at the register-transfer level can be found in [29] and will not be derived in detail here.

Based on the error-masking probability formula, we can analyze static observability of internal signals in HDL designs. First, we construct the control/data flow graph (CDFG) of the HDL design. Then, starting from observation points and analyzing backward along paths in the CDFG, we calculate SOBS values for internal signals according to the formulas shown in Figure 9 [Figure 9: see original paper]. The SOBS values of observation points in the verification process (e.g., primary output variables) are set to 0. When an internal signal a can be observed through multiple paths in the CDFG, its SOBS value takes the minimum among the SOBS values on these paths. For branch statements, the SOBS value of the conditional expression takes the minimum among the SOBS values of signals assigned in its branches.

From this, we know that: (1) larger SOBS values indicate signals are more difficult to observe; (2) when examining a path backward from observation points, SOBS values exhibit monotonically increasing characteristics. Based on these two characteristics, we propose a low-observability source analysis method. This method uses a scoring approach to filter internal observation points, calculating an IOS_score for each internal signal using SOBS values. The internal signal with the highest IOS_score ultimately becomes the ideal internal observation point.

The main steps of this method are as follows:

1. Traverse the CDFG corresponding to the circuit design to obtain the SOBS value for each internal signal. If a signal' s SOBS value exceeds the set threshold T_SOBS, that signal is considered a difficult-to-observe signal.
2. For each difficult-to-observe signal a, analyze along its easiest propagation path l (i.e., the path yielding the minimum calculated SOBS value for a). If the difference between SOBS values of two successive signals on path l is sufficiently large, it indicates that the operation op between these two signals is a significant obstacle to error effect propagation on path l. Therefore, the IOS_score of the signal corresponding to the operand of op is incremented.
3. The internal signals with the highest IOS_score values are ideal candidates for internal observation points.

For a path l, it can be represented as a sequence of successively appearing signals: s , s , ⋯, s , where s is typically the observation point. For two successive signals

s and s on path l, if the difference between their SOBS values exceeds the set threshold *T_SOBS_diff*, it indicates that due to the presence of *op*, the observability of signal s is much worse than its successor signal s . According to the SOBS calculation formula introduced earlier, a signal' s SOBS value on a path is the accumulation of its successor signal' s SOBS value. Therefore, if there are difficult-to-observe signals before s on path l (i.e., among signals s , s , ⋯, s), then s is a cause of their poor observability. Ultimately, if a signal a has a higher IOS_score, it indicates that more difficult-to-observe signals in the circuit are affected by signal a. Thus, using signal a as an internal observation point can solve observability problems for more signals, achieving multiplier effects. More detailed algorithm descriptions and examples can be found in [21][29].

We conducted experiments and analysis on ITC-99 benchmark circuits. For each design, we compared error coverage under three schemes using the same stimulus set: (1) using only output variables as observation points; (2) using output variables as observation points and randomly selecting some difficult-to-observe signals as internal observation points; (3) using output variables as observation points and selecting signals with the highest IOS_score values reported by the low-observability source analysis method as internal observation points. Notably, to ensure fair comparison between schemes 2 and 3, the total number of bits of internal observation points was kept as equivalent as possible during selection.

Figure 10 [Figure 10: see original paper] vividly illustrates the advantages of selecting internal observation points through low-observability source analysis. The vertical axis shows modeled error coverage, and the horizontal axis shows circuit designs. Since internal observation points in scheme 2 are randomly selected from the difficult-to-observe signal set, to avoid selection randomness, scheme 2 was repeated three times with different random selections of difficult-to-observe signals each time (scheme 2 could only be repeated twice for b14 due to limited numbers of difficult-to-observe signals). Scheme 2-1, 2-2, and 2-3 in Figure 10 refer to these three repetitions of scheme 2.

As shown in Figure 10, although randomly selecting different difficult-to-observe signals as internal observation points in scheme 2 can improve error coverage to varying degrees, none reach the effectiveness of scheme 3. This demonstrates that using low-observability sources as internal observation points can solve observability problems for more internal signals at lower cost.

In terms of time overhead, each circuit design in the experiments required only a few seconds to analyze. Some dynamic analysis methods, such as the method in [20], require simulation result analysis after functional simulation to discover difficult-to-observe signals before conducting further simulation. Therefore, using the low-observability source analysis method can analyze and add ideal internal observation points before functional simulation, thus saving verification overhead.

5 Conclusion

Design verification is a critical supporting technology for chip design. By analyzing the dilemmas faced by industry-standard simulation verification technology as design scales develop and their underlying causes, we believe that error model-based design verification technology is a promising direction for breakthrough progress.

This paper elaborates on the research status of design error models. Among these, explicit error models are a highly concerned research area, with mutation testing as the representative work. For design errors involving missing clauses in expressions that cannot be represented by existing error models, we have proposed the item-missing error model and its testing method, which can greatly improve the probability of discovering such design errors.

To date, no widely accepted error model exists in the design verification domain. Against this background, research on error model-related technologies cannot be separated from error injection platforms. This paper introduces our developed error injection system, ErrorInjector. Built upon a compiler foundation, this system supports a rich variety of design error models and features well-designed extensible interfaces based on dynamic link libraries, allowing users to conveniently develop custom error models. This system is highly beneficial for design error model research. In addition to explicit error models, studying design-for-verification techniques from the perspective of effects triggered by design errors is also an excellent approach. We propose a static observability quantification analysis method based on error-masking probability and, building upon this, a method for selecting internal observation points according to low-observability root causes, advancing research in this field.

References

[1] http://public.itrs.net
[2] Bergeron J, *Writing Testbenches Using SystemVerilog*, Springer, 2006
[3] Palnitkar S, *Design Verification with e*, Prentice Hall PTR, 2003.
[4] Yi Jiangfang, Tong Dong, Cheng Xu, "GATEST: A verification platform for automatic generation of simulation vectors using genetic algorithms," *Acta Scientiarum Naturalium Universitatis Pekinensis*, Vol. 42 No. 5, pp. 668-673, 2006.
[5] I. Wagner, V. Bertacco, and T. Austin, "Microprocessor verification via feedback-adjusted Markov models," *IEEE Trans. on CAD*, vol. 26, pp. 1126-1138, Jun. 2007.
[6] Wei Lu, Xiutao Yang, Tao Lv, Xiaowei Li, "An Efficient Evaluation and Vector Generation Method for Observability-Enhanced Statement Coverage," *Journal of Computer Science and Technology*, Vol.20, No.6, Nov., 2005, pp.875-884
[7] http://www.intel.com/support/processors/pentium/fdiv/wp/
[8] Tao Lv, Lingyi Liu, Yang Zhao, Huawei Li, Xiaowei Li, "An observability

branch coverage metric based on dynamic factored use-define chains," *Proc. of IEEE 15th Asian Test Symposium*, Fukuoka, Japan, Nov. 2006, pp.89-94.
[9] R. A. Demillo, R. J. Lipton, F. G. Sayward, "Hints on test data selection: help for the practicing programmer," *IEEE Computer*, pp.34-41, April 1978.
[10] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering Methodology*, 1992, vol. 1, pp. 3-18.
[11] K. S. H. T. Wah, "Fault coupling in finite bijective functions," *The Journal of Software Testing, Verification, and Reliability*, 1995, vol. 5, pp. 3-47.
[12] K. N. King, A. J. Offutt, "A Fortran language system for mutation-based software testing," *Software-Practice & Experience*, Vol.21, No.7, pp.685-718, 1991.
[13] G. Al Hayek, C. Robach, "From specification validation to hardware testing: a unified method," *Proc. of International Test Conference*, pp.885-893, 1996.
[14] D. Barclay, J. Armstrong, "A heuristic chip-level test generation algorithm," *Proc. of Design Automation Conference*, pp.257-262, 1986.
[15] H. AL-Asaad, J. P. Hayes, "Design verification via simulation and automatic test pattern generation," *Proc. of International Conference on Computer-aided Design*, pp.174-180, 1995.
[16] D. Van Campenhout, H. AL-Asaad, J. P. Hayes, T. Mudge, R. B. Brown, "High-level design verification of microprocessors via error modeling," *ACM Trans. on Design Automation of Electronic Systems*, Vol.3, No.4, pp.581-599, Oct. 1998.
[17] Yinghua Min, Stephen Y. H. Su, "Testing functional faults in VLSI," *Proc. of Design Automation Conference*, pp.384-392, 1982.
[18] Li Shen, Stephen Y. H. Su, "A functional testing method for microprocessors," *IEEE Trans. on Computers*, Vol.37, No.10, pp.1288-1293, Oct. 1988.
[19] Katarzyna Radecka, Zeljko Zilic, "Design verification by test vectors and arithmetic transform universal test set," *IEEE Trans. on Computers*, Vol. 53, No.5, pp. 628-640, 2004.
[20] T. Y. Jiang, C. N. J. Liu, and J. Y. Jou, "Observability analysis on HDL descriptions for effective functional validation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol.26, No.8, pp.1509-1521, August, 2007.
[21] Tao Lv, Huawei Li, Xiaowei Li, "Automatic selection of internal observation signals for design verification," *Proc. of IEEE 27th VLSI Test Symposium*, Santa Cruz, USA, May, 2009, pp. 203-208.
[22] Yi Feng, Zheng Zhou, Dong Tong, Xu Cheng, "Clock domain crossing fault model and coverage metric for validation of SoC design," *Proc. of the Conference on Design, Automation and Test in Europe*, pp.1385-1390, 2007.
[23] A. Avizienis, Yutao He, "Microprocessor entomology: a taxonomy of design faults in COTS microprocessors," *Dependable Computing for Critical Applications 7*, pp.3-23, Nov. 1999.
[24] Tao Lv, Tong Xu, Yang Zhao, Huawei Li, Xiaowei Li, "Bug analysis and corresponding error models in real designs," *Proc. of IEEE 12th High Level Design Validation and Test Workshop*, Irvine, USA, Nov. 2007, pp.
[25] Advanced RISC Machines Ltd (ARM), *AMBA AHB Specification (Revision*

2.0), ARM IHI 0011A, http://www.arm.com/products/solutions/AMBA_Spec.html
[26] http://www.icarus.com/eda/verilog/
[27] D. Van Campenhout, T. Mudge, J. P. Hayes, "Collection and analysis of microprocessor design errors," *IEEE Design & Test of Computers*, Vol.17, No.4, pp.51-60, Oct.-Dec. 2000.
[28] M. N. Velev, "Collection of high-level microprocessor bugs from formal verification of pipelined and superscalar designs," *Proc. of International Test Conference*, pp.138-147, 2003.
[29] Tao Lv, *Research on Quantitative Evaluation Methods for Digital Integrated Circuit Design Verification*, Ph.D. dissertation, Graduate University of Chinese Academy of Sciences, Beijing, 2009
[30] Wang Tiancheng, *Injection Methods and Implementation of Design Errors in Digital Integrated Circuits*, Master' s thesis, Graduate University of Chinese Academy of Sciences, 2009.

Author Biographies

Tao Lv: Assistant Researcher, Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, lvtao@ict.ac.cn
Huawei Li: Professor and Ph.D. Supervisor, Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences
Xiaowei Li: Professor and Ph.D. Supervisor, Institute of Computing Technology, Chinese Academy of Sciences; Deputy Director, Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences; Council Member, China Computer Federation; Director, Fault-Tolerant Computing Professional Committee; Associate Editor, JCST

---

[1] Part of the content of this paper has appeared in references [29] and [30].

Note: Figure translations are in progress. See original paper for figures.