Evaluation of Design Error Models for Verification Testing of Microprocessors[1]

David Van Campenhout, Trevor Mudge and John P. Hayes

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122
E-mail: {davidvc, jhayes, tnm}@eecs.umich.edu

ABSTRACT

We are developing a design verification methodology for microprocessor hardware based on modeling design errors and generating simulation vectors for the modeled errors via physical fault testing techniques. A class of conditional error models has recently been proposed for use with this methodology and is the subject of this paper. We analyze the classes of errors that are guaranteed to be detected by complete test sets for the conditional errors for gate level circuits. We also report experimental work-in-progress aimed at comparing the usefulness of different design error models for verification testing of microprocessors. In these experiments the coverage of test sets for both synthetic and actual errors is computed.

1. INTRODUCTION

Design verification is an essential activity in the design of high-performance microprocessors, especially for safety-critical applications. The goal of design verification is to ensure the functional correctness of a design (implementation) with respect to its specification.

Simulation-based design verification tries to uncover design errors by detecting a circuit's faulty behavior when deterministic or pseudo-random tests (simulation vectors) are applied. Manufacturers of microprocessors still rely heavily on simulation-based methods to verify their products. An important issue is that of quantifying the effectiveness of these methods. A variety of coverage metrics have been proposed: code coverage from software testing [2, 16], finite state machine based metrics [10, 9, 7, 13], an observability-based metric [5, 6], and design-specific metrics such as architectural events [12, 14]. A shortcoming of these metrics is that the relationship between a metric and the classes of design errors that they detect is not well understood.

An alternative approach draws on the similarity between hardware design verification and physical fault testing. In this approach [1, 11, 3, 4, 15] synthetic error models are derived from empirical design error data, and physical fault testing techniques are adapted to generate test sets targeted at the synthetic errors. The methodology is illustrated in Figure 1. Given are a design implementation, typically described at the RTL level, a design specification, typically an instruction set architecture (ISA) model of the processor. Both models are assumed to be executable. A synthetic design error

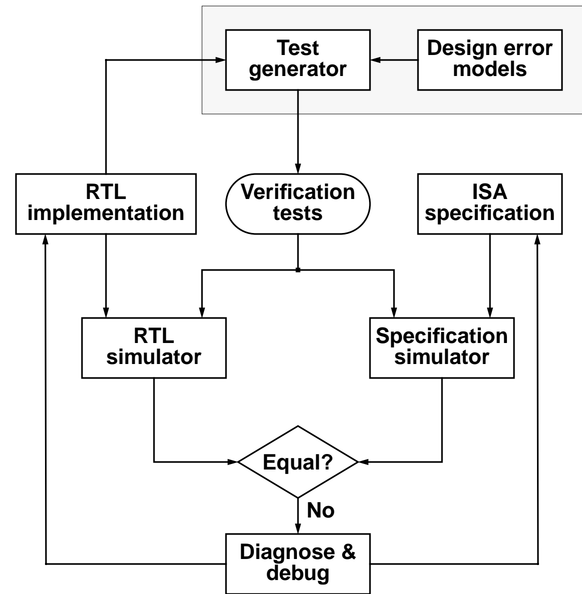

FIGURE 1. Simulation-based hardware design verification via error modeling

model is selected and a test set is generated that provides complete coverage of modeled errors in the implementation. Both the implementation and its specification are simulated for that test set. A discrepancy in the simulation outcome indicates an error in the design implementation or in its specification.

Some important features of the approach are as follows. The methodology can guarantee complete coverage of specific classes of design errors. The approach addresses the issue of observability. Test are generated that excite errors and propagate an error effect to the primary outputs so that a discrepancy will be detected by comparing the simulation outcome against that of the specification. A limitation of the approach is that sequential test generation for very large circuits such as microprocessors is a hard problem. The sequential nature of the problem is due to the gap in abstraction between the specification and the implementation, and in contrast to physical fault testing, design-for-testability techniques do not apply.

Section 2 presents the conditional error model. In section 3 its relationship to other error models is analyzed. Section 4 presents error simulation experiments, and concluding remarks are given in Section 5.

1. This research is supported by DARPA under Contract No. DABT63-96-C-0074. The results presented herein do not necessarily reflect the position or the policy of the U.S. Government.

2. DESIGN ERROR MODELS

To be useful for design verification, error models should satisfy three requirements: (1) tests (simulation vectors) that are complete for the modeled errors should also provide very high coverage of actual design errors; (2) the modeled errors should be amenable to automated test generation; (3) the number of modeled errors should be relatively small. In practice, the third requirement means that error models that define a number of error instances linear, or at most quadratic in the size of the circuit are preferred. The error models need not mimic actual design bugs precisely, but the tests derived from complete coverage of modeled errors should provide very good coverage of actual design bugs.

A set of error models that satisfy the requirements for the restricted case of gate-level logic circuits was developed in [3]. Motivated by empirical design error data, similar error models for higher-level (RTL) designs were proposed in [15]. Five basic error models were proposed: 1) bus single-stuck line error (SSL) 2) module substitution error, 3) bus order error, 4) bus source error, and 5) bus driver error.

Experiments described in [15] show that complete test set for these basic errors can uncover many actual design errors. To increase coverage of actual errors to the very high levels needed for design verification, additional error models are required to guide test generation. Design error data and error occurrence statistics can be used to construct additional error models. Although, this extended set of error models increases the number of actual errors that can be modeled directly, they are too complex for practical use in manual or automated test generation.

Analysis of the more difficult actual errors revealed that these errors are often composed of multiple basic errors, and that the component basic errors interact in such a way that a test to detect the actual error must be much more specific than a test to detect any of the component basic errors. An effective error model should necessitate the generation of these more specific tests without resorting to direct modeling of the composite errors. The complexity of the new error models should be comparable to that of the basic error models and the (unavoidable) increase in the number of error instances should be controlled to allow trade-offs between test generation effort and verification confidence. These requirements can be combined by augmenting the basic error models with a condition.

A conditional error (C,E) consists of a condition C and a basic error E; its interpretation is that E is only active when C is satisfied. In general, C is a predicate over the signals in the circuit during some time period. To limit the number of error instances, we restrict C to a conjunction of terms $(y_i = w_i)$ where y_i is a signal in the circuit and w_i is a constant of the same bit-width as y_i and whose value is either all-0s or all-1s. The number of terms (condition variables) appearing in C is said to be the order of (C,E). Specifically, we consider the following conditional error (*CE*) types:

- Conditional single-stuck line (CSSLn) error of order n;
- Conditional bus order error (CBOEn) of order n;
- Conditional bus source error (CBSEn) of order n.

When $n = 0$, a conditional error (C,E) reduces to the basic error E from which it is derived. Higher-order conditional errors enable the generation of more specific tests, but lead to a greater test generation cost due to the larger number of error instances. For example, the number of CSSLn errors in a circuit with N signals is $2^{n+1}N^{n+1}$. Although the total set of all N signals we consider for each term in the condition can possibly be reduced, CSSLn errors where $n > 1$ are probably not practical. In the remainder of the paper we restrict our focus to CSSL1 errors.

3. ANALYTICAL EVALUATION OF CSSL1 MODEL

In this section we analyze the detection of basic design errors by complete test sets for CSSL1 errors for gate level circuits. Let D_0 be a gate-level circuit; construct D_1 by injecting a single error e_1 into D_0, where e_1 is an instance of error model M_1. Let T_0 (T_1) be a test set that provides complete coverage of all detectable CSSL0 (CSSL1) errors in D_1. We analyze the coverage provided by test sets T_0 and T_1 with respect to the error models M_1 proposed in [3]. In particular we are interested in those error classes covered by T_1, but not by T_0.

We use the notation introduced in [3], and refer the reader to that paper for further details of the error models. Let $y = G(x_1,\ldots,x_n)$ be a gate in the error-free circuit. A gate substitution error G/G' occurs if a gate G is erroneously replaced with gate G' that has the same number of inputs but is from a different type. The set of all 2^n input vectors of an n-input gate is divided into disjoint subsets V_0, $V_1,\ldots,V_n$, where V_k contains all input vectors with exactly k 1s in their binary representation, $0 \le k \le n$. The disjoint sets V_{null}, V_{all}, V_{odd}, and V_{even} are defined as follows:

$V_{null} = V_0$; $V_{all} = V_n$;

$V_{odd} = \bigcup_{i = odd \wedge i \ne n} V_i$; $\bigcup_{i = even \wedge i \ne 0 \wedge i \ne n} V_i$.

The sets V_{null}, V_{all}, V_{odd}, and V_{even} are called the *characterizing sets* or *C-sets* of G.

Consider the following sets of CSSL0 and CSSL1 errors (in the erroneous circuit):

- $E_1 = \{(1, y/0)\}$
- $E_2 = \{(1, y/1)\}$
- $E_3 = \{(1, x_i/0) \mid i=1\ldots n\}$
- $E_4 = \{(1, x_i/1) \mid i=1\ldots n\}$
- $E_5 = \{(x_i = 0, y/0) \mid i=1\ldots n\}$
- $E_6 = \{(x_i = 1, y/0) \mid i=1\ldots n\}$
- $E_7 = \{(x_i = 0, y/1) \mid i=1\ldots n\}$
- $E_8 = \{(x_i = 1, y/1) \mid i=1\ldots n\}$

Let T_i, where $i = 1\ldots8$, be a complete test set for E_i.

Single-input gate substitution errors

Gate substitution errors (*GSE*s) involving buffers or inverters, are called *single-input GSE*s. A necessary and sufficient condition to detect G/G' is to sensitize y. Any test in T_1 or T_2 must sensitize y and hence detects G/G'. If both T_1 and T_2 are empty, y is not sensitizable and hence G/G' is undetectable.

Multiple-input GSEs

Consider a *multiple-input GSE* (*MIGSE*) G,AND. To detect a (G,G'), we have to identify G' in the erroneous circuit. Note that:

- any test in T_1 or T_3 or T_6 excites G' for V_{all};
- any test in T_2 or T_7 excites G' for $V_{null} \cup V_{even} \cup V_{odd}$;
- any test in T_4 excites G' for V_{n-1}.
- any test in T_8 excites G' for $V_{even} \cup V_{odd}$.

- $T_5 = \emptyset$

Case 1: $T_4 \neq \emptyset$, $T_1 \neq \emptyset$

$T_1 \cup T_4$ uncovers all detectable *MIGSEs* with the exception of XNOR/AND (XOR/AND) for n even (odd). Detection of this last error requires exciting G' for $V_{null} \cup V_{even}$ ($V_{null} \cup V_{odd}$). None of the error sets considered can enforce this condition.

Case 2: $T_4 \neq \emptyset$, $T_1 = \emptyset$

T_4 uncovers all detectable *MIGSEs* with the exception of NOR/AND and XNOR/AND (XOR/AND) for n even (odd). Detection of these last two errors requires exciting G' for both V_{null} and V_{even} (V_{odd}). None of the error sets considered can enforce this condition.

Case 3: $T_4 = \emptyset$, $T_8 \neq \emptyset$, $T_1 \neq \emptyset$

$T_1 \cup T_8$ uncovers all detectable *MIGSEs* with the exception of XNOR/AND (XOR/AND) for n even (odd). $T_1 \cup T_2$ might fail to uncover OR/AND as well.

Case 4: $T_4 = \emptyset$, $T_8 \neq \emptyset$, $T_1 = \emptyset$

T_8 might fail to detect NOR/AND and either XNOR/AND or XOR/AND.

Case 5: $T_4 = \emptyset$, $T_8 = \emptyset$

G' cannot be excited for $V_{even} \cup V_{odd}$. $T_1 \cup T_2$ excites G' for each remaining C-set.

The analysis for *MIGSEs* where G'=NAND,OR,NOR,XOR,XNOR is similar to that presented above.

We conclude that the coverage of *MIGSEs* provided by complete test sets for CSSL1 errors is only marginally better (see case 3) than that provided by test complete test sets for CSSL0 errors.

Gate count errors

Two types of gate count errors are defined in [3]: extra-gate errors and missing gate errors. Extra-gate errors were shown to be detected by a complete test set for *GSEs*. Hence the same conclusions with respect to coverage by complete test sets apply to CSSL1 errors.

It was shown in [3] that detection of missing gate errors requires applying either V_2 or V_{n-2}. These sets cannot be enforced using a single CSSL1 error, and hence complete test sets for CSSL1 errors do not provide increased coverage for missing gate errors than test sets for CSSL0 errors.

Input count errors

Extra input errors were shown to be covered by complete test set for CSSL0 errors [3]. The coverage for missing input errors was shown to be only partial. A missing input error occurs when an n-input gate ($n \geq 3$) is replaced by an ($n-1$)-input gate with its $n-1$ inputs connected to an arbitrary subset of the original n inputs. The error detection requirements of this type of error map exactly on those of a CSSL1 error, except for XOR or XNOR gates.

Wrong input error

A wrong input error occurs when a single gate input is connected to the wrong signal: in the error-free circuit, $y = G(x_1,\ldots,x_n)$, while in the erroneous circuit: $y = G(z, x_2,\ldots,x_n)$. A necessary and sufficient condition to detect this error is to sensitize z while z and x_1 have opposite values. This is equivalent to detecting either of the CSSL1s ($x_1 = 0$, z/0) or ($x_1 = 1$, z/1).

Missing 2-input gate error

This error occurs if the error-free circuit contains a gate $y = G(x_1, x_2)$ that is completely missing in the erroneous circuit: $y = x_1$. It can be shown that the error detection requirements for this error are equivalent to those of a CSSL1 error. For example if G=AND, the corresponding CSSL1 error is ($x_2 = 0$, y/0). A complete test set CSSL0 errors may fail to detect this error.

Discussion

Complete test sets for CSSL1 errors will also detect all wrong input errors, all missing input errors on gates that are not of type {XOR,XNOR}, and all missing 2-input gate errors; complete test sets for CSSL0 errors may fail to detect these errors. For the other error types, no increased coverage is guaranteed by complete test sets for CSSL1 errors.

4. EXPERIMENTAL EVALUATION OF CSSL1 MODEL

From the analysis presented in the previous section, one could conclude that the class of design errors which is guaranteed to be detected when using a test set that is complete from CSSL1 errors is very limited, and in fact most actual design errors do not fall in this class.

However, a case study presented in [15], demonstrates that for practical cases, test sets that are complete for CSSL1 errors can be very effective at detecting actual design errors. During the debug process of a microprocessor design, design errors were systematically recorded. Each actual error was examined and it was determined if a modeled design error exists that is dominated by the actual error. Such a dominated modeled error has the property that any test that detects it will also detect the actual error. The case study showed that by targeting the CSSL1 model the coverage of actual errors can be significantly improved over that provided by test sets complete for only basic error models such as SSL errors.

An alternative method to compare the effectiveness of two design error models is to take an unverified design, and generate test sets that are complete with respect to the two error models. The test set that uncovers more (and harder) design errors in a fixed amount of time is more effective. However, for such a comparison to be practical, fast and efficient high-level test generation tools for our error models appear to be necessary. Although this type of test generation is feasible, it has yet to be automated. Instead we consider test sets that were not specifically targeted, and compute their coverage of modeled design errors as well as of actual design errors.

We are presently conducting a set of experiments whose goal is to compare different design error models and investigate the relationship between coverage of modeled design errors and coverage of (more complex) actual errors.

The test vehicle for this study is the well-known DLX microprocessor [8]. The particular DLX version considered is a student-written design that implements 44 instructions, has a five-stage pipeline and branch prediction logic, and consists of 1552 lines of structural Verilog code, excluding the models for library modules such as adders, registerfiles, etc. The design errors made by the student during the design process were systematically recorded. Some characteristics of two of the modules of the design are shown in Table 1. Module `top` integrates the different pipeline stages and contains

TABLE 1. Characteristics of two modules of the DLX microprocessor implementation.

Parameter	module 1: `top`	module2: `decode`
# Lines of code	302	263
# CSSL0 errors	574	816
# CSSL1 errors	141,756	238,732
# Restricted CSSL0 errors	178	82
# Restricted CSSL1 errors	21,864	18,788
# Detectable actual errors	8	16

TABLE 2. Coverage [%] of synthetic and actual errors by biased random test sets T0-T13.

Error set	module 1: `top`	module2: `decode`
CSSL0 errors	77	64
Restricted CSSL0 errors	86	93
Restricted CSSL1 errors	72	69
Actual errors	75	69

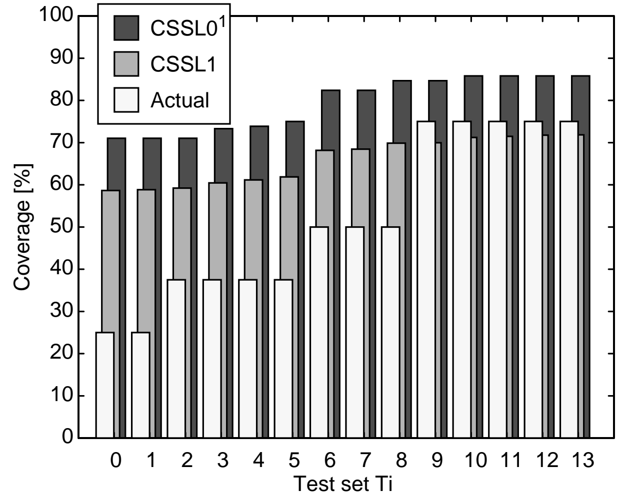

FIGURE 2. Coverage [%] of restricted CSSL0, restricted CSSL1, and actual errors by 14 biased random test sets T0-T13 for `top`.

1. The three bars at abscissa *i* represent the coverage of restricted CSSL0 errors, restricted CSSL1 errors, and actual errors in module `top` by test programs T0 ∪ T1 ∪…T*i*.

the forwarding logic. Module `decode` describes the decode stage of the pipeline. These modules are presented here because 75% of all actual errors were made within these two modules.

Efficient error simulation tools for conditional error models are yet to be developed. As a temporary solution, we modified the design description to allow us to inject conditional errors into the design. The modifications do not cause a significant overhead during simulation and do not require recompilation of the simulator when a new error is injected. On the other hand, this approach requires a simulation run for each error considered.

The error models considered in this study are the CSSL0 and CSSL1 models. Even for these moderately sized modules, the number of CSSL1 errors is very large; for example, there are 141,756 CSSL1 errors in `top`. Given our error simulation approach, the number of errors needs to be reduced to make the experiment practical. A subset of the CSSL1 errors was selected by imposing the following constraints: 1) lines considered in the predicate of the conditions are restricted to signals of bit width 1, and 2) lines considered as error sites are restricted to signals with bit width > 1. *CEs* of this type are referred to as *restricted CEs*. This reduces the number of CSSL1 errors by about an order of magnitude. There are 21,864 restricted CSSL1 errors in `top`. Error simulation for restricted CSSL1 errors in `top` and the test set described below took 34 hours on a HAL300 workstation; an average simulation speed of 140 simulated clock cycles per CPU second was observed.

We developed a tool for generating random (but valid) assembly programs for the DLX instruction set architecture. The tool is biased towards generating "interesting" cases, such as data dependencies, control dependencies, exceptions, and boundary data values. To satisfy the requirement that the programs generated be valid, some structure is imposed on the programs which limits the variety of the programs. A number of parameters allow the user to vary the size and structure of the programs. We constructed a sequence of test programs increasing in size and complexity. The combined execution length of the programs was 3445 cycles.

We then computed the coverage for synthetic and actual errors of the test set. The results are summarized in Table 2. A significant fraction of CSSL0 errors is not covered by the test set. The coverage of restricted CSSL1 errors is, as expected, lower than that of restricted CSSL0 errors. Figure 2 shows the coverage of three error sets as a function of the number of test programs applied. The first (and shortest) test program uncovers many "easy" errors. Coverage increases slowly as more test programs are applied. The profiles for both synthetic error models appear similar. Both error models reveal areas insufficiently exercised by the test programs. For instance, none of the test programs contained instructions with illegal opcodes.

5. DISCUSSION

Our analytical evaluation of the CSSL1 error model shows that complete test sets for CSSL1 errors also detect a number basic errors for which complete test sets for CSSL0 errors provide only partial coverage. However, as the CSSL1 error model defines a number of error instances quadratic in the size of the circuit, further evidence is needed to demonstrate the model's merit for design verification. We therefore have been conducting error simulation experiments in which the coverage of CSSL0 and CSSL1 errors by biased random test sets is computed. So far these experiments do not indicate that there is better correlation between coverage of the synthetic errors and coverage of actual errors for one or the other synthetic error model. Additional experiments using complete test sets for the synthetic error models will be necessary to assess the relative effectiveness of the CSSL1 and the CSSL0 model.

ACKNOWLEDGMENTS

We thank Steve Raasch for providing his DLX design.

REFERENCES

[1] M. S. Abadir, J. Ferguson, and T. E. Kirkland. “Logic design verification via test generation.” *IEEE Trans. Computer-Aided Design*, vol. 7, no. 1, pp. 138–148, January 1988.

[2] A. Aharon, A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, and V. Schwartzburd. “Verification of the IBM RISC System/6000 by dynamic biased pseudo-random test program generator.” *IBM Systems Journal*, pp. 527–538, 1991.

[3] H. Al-Asaad and J. P. Hayes. “Design verification via simulation and automatic test pattern generation.” In *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 174–180.

[4] H. Al-Asaad, D. Van Campenhout, J. P. Hayes, T. Mudge, and R. B. Brown. “High-level design verification of microprocessors via error modeling.” In *Dig. of IEEE Int. High Level Design Validation and Test Workshop*, 1997, pp. 194–201.

[5] S. Devadas, A. Ghosh, and K. Keutzer. “Observability-based code coverage metric for functional simulation.” In *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 418–425.

[6] F. Fallah, S. Devadas, and K. Keutzer. “OCCOM: Efficient computation of observability-based code coverage metric for functional simulation.” In *Proc. Design Automation Conf.*, 1998, pp. 152–157.

[7] A. Gupta, S. Malik, and P. Ashar. “Toward formalizing a validation methodology using simulation coverage.” In *Proc. Design Automation Conf.*, 1997, pp. 740–745.

[8] J. Hennessy and D. Patterson. *Computer Architecture: A quantitative Approach*. Morgan Kaufman Publishers, San Mateo, Calif., 1990.

[9] R. C. Ho and M. A. Horowitz. “Validation coverage analysis for complex digital designs.” In *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 146–151.

[10] R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill. “Architecture validation for processors.” In *Proc. Int. Symp. Computer Architecture*, 1995, pp. 404–413.

[11] S. Kang and S. A. Szygenda. “The simulation automation system SAS; concepts, implementations, and results.” *IEEE Trans. on VLSI*, 1994.

[12] M. Kantrowitz and L. M. Noack. “I’m Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha microprocessor.” In *Proc. Design Automation Conf.*, 1996, pp. 325–330.

[13] D. Moundanos, J. A. Abraham, and Y. V. Hoskote. “Abstraction techniques for validation coverage analysis and test generation.” *IEEE Trans. Computers*, vol. 47, no. 1, pp. 2–14, January 1998.

[14] S. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey. “Functional verification of a multiple-issue, out-of-order, superscalar Alpha processor - the DEC Alpha 21264 microprocessor.” In *Proc. Design Automation Conf.*, 1998, pp. 638–643.

[15] D. Van Campenhout, H. Al-Asaad, J. P. Hayes, T. Mudge, and R. B. Brown. “High-level design verification of microprocessors via error modeling.” to appear in *ACM Trans. Design Automation of Electronic Systems*, vol. 3, no. 4, October 1998.

[16] R. Vemuri and R. Kalyanaraman. “Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming.” *IEEE Trans. on VLSI*, pp. 201–214, 1995.