A Method for the Evaluation of Behavioral Fault Models*

Emilio Gaudette, Michael Moussa
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
elmilio2@yahoo.com, ed_b_tz@yahoo.com

Ian G. Harris
Department of Computer Science
University of California Irvine
Irvine, CA 92697
harris@ics.uci.edu

Abstract

Many fault models have been proposed which attempt to capture design errors in behavioral descriptions, but these fault models have never been quantitatively evaluated. The essential question which must be answered about any fault model is, "If all faults in this model are detected, is the design guaranteed to be correct?" In this paper we present a method to examine the degree to which an arbitrary fault model can ensure the detection of all design errors. The method involves comparing fault coverage to *error coverage* as defined by a practical design error model which we describe. We have employed our method to perform a limited analysis of the statement and branch coverage fault models.

1 Introduction

Validation techniques verify functionality by simulating (or emulating) a system description with a given test input sequence. The validation process is known to be a cost bottleneck and time bottleneck in the overall design process. A large component of validation cost is the test generation process required to ensure the detection of all design errors. The cost of the test generation process derives from the largely manual nature of the process, necessitating the effort of multiple designers for an extended period of time. Automation of the test generation process is essential to greatly reduce time to market and design cost. A key component of automatic test generation is the fault model which abstractly describes the expected erroneous behaviors. Fault models are needed to provide detection goals for the automatic test generation process, and fault models enable the error detection qualities of a test sequence to be evaluated. The chief benefit of using a fault model is that the detection of a small number of faults ensures the detection of a large number of potential design errors. The total set of potential design errors is far too large to target directly during test generation, but the use of fault models makes the test generation process tractable.

The effectiveness of automatic test generation depends on the effectiveness of the underlying fault models used. If the detection of all faults in a fault model does not guarantee the detection of the majority design errors, then test generation is not complete. Automatic test generation tools cannot be reliably used for design validation until the set of fault models used have been evaluated and are known to ensure the detection of nearly all design errors. Many fault models have been identified in previous research [3] but the ability of these fault models to ensure detection of real design errors has never been evaluated.

We have developed a technique to evaluate the design error detection ability of an arbitrary behavioral fault model. Fault coverage, for a given test sequence and design, is compared to error coverage based on a design error model which we present. The design error model which we use is known to capture a significant subset of real design errors. By comparing fault coverage to error coverage for many test sets and benchmark designs, it is possible to draw some statistical conclusions about the effectiveness of the fault model. We have used our proposed method to perform a limited evaluation of the statement and branch coverage fault models.

The remainder of this paper is organized as follows. Previous research in fault modeling for validation is summarized in Section 2. The relationship between fault models and design errors is described in Section 3. The details of the evaluation method are presented in Section 4 and the set of design errors which we consider is described in Section 5. The evaluation of statement and branch coverage is described in Section 6, and

*This work was supported in part by the National Science Foundation under Grant No. 0204134

conclusions are presented in Section 7.

2 Previous Work

Several fault models have been developed to evaluate behavioral designs, many of which are based on software test fault models. A survey of fault models used for validation of behavioral hardware and software descriptions can be found in [3]. A number of fault models are based on the traversal of paths through the control dataflow graph (CDFG) representing the system behavior. The earliest control-dataflow fault models include statement coverage and branch coverage [1] models used in software testing. Statement coverage associates a potential fault with each line of code, and requires that each statement in the description be executed during testing. The branch coverage metric associates potential faults with each direction of each conditional in the CDFG.

Although significant research has been performed in the development of fault models, the evaluation of these models in terms of design error coverage has not been convincing. Fault models are either compared to other fault models which also have not been evaluated, or only anecdotal evidence is used to justify their usefulness. To our knowledge, no fault model has been shown to cover any class of real design errors.

3 Fault Models and Design Errors

The central goal of this research is to evaluate fault models by understanding the relationship between a fault model and the set of real design errors. In order to more clearly describe the problem, some definitions are required. A design error is a difference between the abstract design concept and the executable design description. Design errors may range from simple syntax errors, confined to a single line of a design description, to a fundamental misunderstanding of the design specification which may impact a large segment of the description. The goal of design validation is to identify all potential design errors through simulation. The number of potential design errors is too large to be managed either automatically or manually, so a method is needed to reduce this complexity. A design fault describes the behavior of a set of design errors, allowing a large set of design defects to be modeled by a small set of design faults. A fault model describes the detection criteria of a set of faults for an arbitrary design. A fault model enables the concise representation of the set of all design errors for the purposes of test generation and test quality evaluation.

Figure 1 depicts the relationship between design faults and design errors. The figure shows two sets, the set of all potential faults in a design, and the set of all potential errors in a design. Each fault is said to model a set of errors if the detection of the fault ensures the detection of all of the errors in the set. The errors modeled by faults are indicated using dotted lines in Figure 1. An error is referred to as an unmodeled error if it is not modeled by any fault. In order to rely upon a fault model, the union of all faults must model the vast majority of design errors.

4 Evaluation of Fault Models

A fault model is used to estimate the error detection ability of a test sequence t for a design d by computing a Fault Coverage (FC(d,t)). The FC(d,t) value indicates the degree to which design errors in design d will be detected using test sequence t. For example, a given test sequence may only produce FC(d,t) = 90% using the statement coverage fault model, indicating that only 90% of all statements are executed during simulation with the sequence.

A test sequence t applied to a design d can also be associated with an Error Coverage (EC(d,t)) which indicates the fraction of potential errors in design d which would be detected by test sequence t. For the purposes of our analysis, we assume that a design error is detected if it produces an incorrect result at the system outputs. The number of potential errors in a design is huge, so computation of the EC(d,t) value as part of the design cycle is too costly in practice.

The main purpose of a fault model is to approximate the error detection qualities of a test sequence for a design using the FC(d,t) value. The EC(d,t) value is the true indicator of the error detection abilities of a test sequence by definition, but since it cannot be computed during design, the FC(d,t) value is used as an approximation. A fault model is said to have high accuracy if the difference between FC(d,t) and EC(d,t) is known to be small for almost all designs and test sequences.

We estimate the accuracy of fault models experimentally by computing and comparing the FC(d,t) and EC(d,t) for a wide range of designs and test sequences. Computation of the EC(d,t) is very time consuming but it is necessary to determine the accuracy of a fault

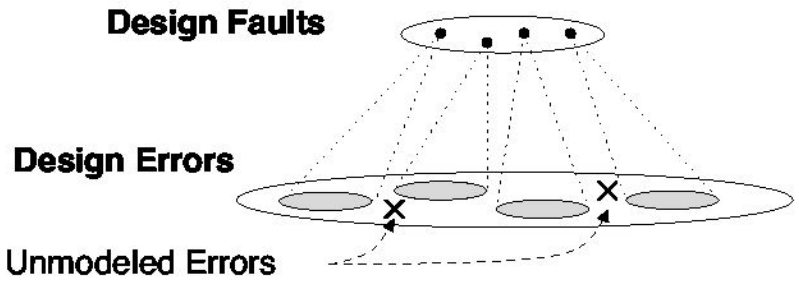

Figure 1: Mapping Between Design Faults and Errors

model. A fault model has perfect accuracy if the quantity FC(d,t) - EC(d,t) is always equal to zero. To estimate the accuracy we compute both the average and the standard deviation of the quantity FC(d,t) - EC(d,t) for a large set of random test sequences. Together the average and standard deviations of FC(d,t) - EC(d,t) reflect how close fault coverage is to true error coverage, and how consistently the fault coverage correlates to error coverage.

5 Design Error Model

Computation of the EC(d,t) requires that each potential design error be inserted into the design individually, and that the erroneous designs be simulated with the test sequence. Inserting errors into a design requires the use of a design error model which describes the set of design errors to be considered. The wide variety of potential design errors makes it impossible to capture all of these errors at this time. Instead, we restrict our investigation to a subset of design errors which has been found to be most common in hardware design [2]. These errors (referred to in [2] as "goof" errors) include simple typographical mistakes and accounted for 12.7% of the design errors found in the Pentium 4. To determine the EC(d,t) value we use the mutation analysis technique studied previously in software testing and hardware validation [4, 5]. In mutation analysis terminology, a mutant is a version of a behavioral description which differs from the original by a single potential design error. A mutation operator is a function which is applied to the original program to generate a mutant. A set of mutation operators describes all expected design errors. The mutation operations which we use are described below.

- Arithmetic Operator Replacement (AOR) - Each occurrence of one of the operators +, -, * and / is replaced by each of the other operators. In addition, each is replaced by the operators LEFTOP and RIGHTOP. LEFTOP returns the left operand; RIGHTOP returns the right operand.
- Relational Operator Replacement (ROR) - Each occurrence of one of the relational operators ($<$, $>$, $\leq$. $\geq$, $=$, $\neq$) is replaced by each one of the other operators. In addition, the expression is replaced by FALSEOP and TRUEOP.
- Variable Replacement (VR) - Each variable in a description unit is replaced by every variable of the same type in the description.

6 Experimental Results

We have applied our method to evaluate the statement coverage and branch coverage fault models. We have used three design benchmarks to perform the evaluation, the greatest common divisor (GCD), the differential equation solver (diffeq) and the traffic light controller (TLC). We have described each benchmark as a Java program and we have inserted design errors into these programs. Analysis is performed with 20 different randomly generated test sequences for each benchmark. The number of test patterns in each test sequence is different for each benchmark and is set to ensure high fault coverage ($> 80\%$) without allowing fault coverage values to consistently saturate at 100%. High fault coverage values are required for evaluation because high fault coverage values are the most likely requirement in practice.

Table 1 shows the basic information about each benchmark, including the number of statements, branches, and design errors using our design error

	Stmts	Brnch	Errors
GCD	19	1	162
Diffeq	21	7	502
TLC	65	14	214

Table 1: Benchmark Information

	SC-EC		BC-EC	
	Avg	Stdev	Avg	Stdev
GCD	16.35	3.17	16.68	3.16
Diffeq	8.88	11.59	14.60	20.45
TLC	10.72	5.06	2.28	6.34

Table 2: FC-EC Results

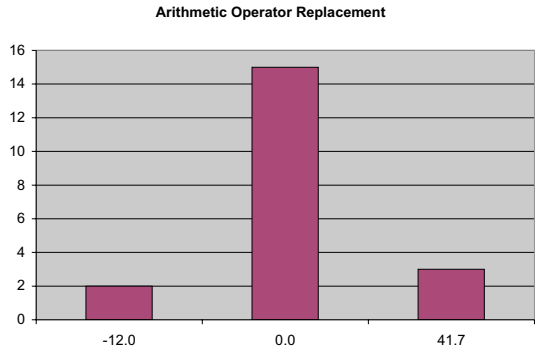

Figure 2: Statement Coverage - Error Coverage, GCD Benchmark

model. Table 2 shows the FC-EC evaluation results. The second and third columns labeled SC-EC show the average and standard deviations of the FC-EC quantity for statement coverage, and the next two columns show the result data for the branch coverage fault model. The numbers in Table 2 are in percent. It is difficult to draw conclusions from this initial experiment which uses only three small design benchmarks, but some interesting trends can still be identified in this experiment. For example, the standard deviation of the FC-EC values is roughly proportional to the number of errors in the design. If this trend continues for larger examples then both statement and branch coverage fault models will be completely unreliable for large examples.

By examining results in more detail it is possible to evaluate the accuracy of a fault model for different types of design errors. Figure 2 shows the distribution of the FC-EC results when statement coverage is used as the fault model and Arithmetic Operator Replacement is used as the error model. The figure shows a distribution of 20 random test runs. The majority of the runs result in a perfect match between statement coverage and error coverage. When there is a difference between statement coverage and error coverage it is usually positive, showing that the statement coverage metric is generally *optimistic* because it overestimates error coverage. The negative differences occur when statement coverage is *pessimistic*. In this case, pessimism could result from the existince of lines of code which contain no arithmetic operators. It would then be possible to achieve 100% coverage for Arithmetic Operator Replacement without achieving 100% statement coverage.

7 Conclusions

We have presented a method to evaluate a fault model by comparing its fault coverage to error coverage based on a subset of design errors. This type of evaluation is essential if new fault models are to be accepted and used in an industrial setting. The analysis which we present will be expanded on in the future to reveal detailed strengths and weaknesses of existing fault models, and to provide direction for the development of new fault models in the future.

References

[1] B. Beizer. *Software Testing Techniques, Second Edition.* Van Nostrand Reinhold, 1990.

[2] B. Bentley. Validating the intel pentium 4 microprocessor. In *Design Automation Conference*, 2001.

[3] I. G. Harris. Hardware-software covalidation: Fault models and test generation. *IEEE Design and Test of Computers*, 20(4):40–47, July-August 2003.

[4] G. Al Hayek and C. Robach. From specification validation to hardware testing: A unified method. In *International Test Conference*, pages 885–893, October 1996.

[5] K. N. King and A. J. Offutt. A fortran language system for mutation-based software testing. *Software Practice and Engineering*, 21(7):685–718, 1991.