Mutation Analysis for the Evaluation of Functional Fault Models

Qiushuang Zhang and Ian Harris
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
E-mail: qzhang@ecs.umass.edu, harris@ecs.umass.edu

I. Introduction

Design validation by simulation-based techniques is the most common approach to verification due to the computational complexity of more formal techniques. Validation entails the generation of a test pattern sequence which is applied to the design during simulation to trigger erroneous behavior. Since simulation can only be performed with a small subset of the entire space of test sequences, some method is needed to estimate the degree of verification achieved by a given test sequence. The degree of verification afforded by a test sequence must be known in order to direct test pattern generation, and to provide the designer with the knowledge that verification goals have been achieved.

Several researchers have proposed different *functional fault models* as tools in determining the degree of verification achieved by a test sequence. A functional fault model describes the space of erroneous design behaviors which can be expected as a result of a design error. A functional fault model is an abstraction of the space of all possible design errors which could be inadvertently created by the designer. Several functional fault models have been developed to capture faults in a behavioral hardware description description [3]. The similarity between the problem of behavioral hardware verification and the problem of software testing has led to the investigation of software metrics as well, including statement, branch, and path coverage [1].

As new fault models are proposed, it is essential that the quality of these fault models be measured before designers can rely on the models for verification. The quality of a functional fault model can be measured by two parameters: (1) the accuracy with which it models the space of all possible design errors, and (2) the number of faults included in the fault model. The size of a given fault model is easily evaluated, but a reliable technique is needed to evaluate the accuracy of the fault model. The problem of measuring the accuracy of a functional fault model for verification is analogous to the problem of relating stuck-at fault coverage to defect coverage for manufacture test [2]. In both problems, an abstract high-level fault model must be evaluated by comparison to a low-level defect model. The defect model is tied directly to the source of the defects, while the fault model is an abstraction of the behavior caused by defects. In the case of manufacture test, the defect model is largely understood because the physical source of spot defects has been studied by previous researchers. An analogous *validation defect model* is more difficult to describe because the source of design defects is a human designer rather than a physical environment. In order to evaluate the accuracy of a functional fault model, it is first necessary to define a validation defect model which describes the design errors most likely to be created by a designer. Validation defect models have been proposed previously for gate-level circuits [5], but little work has been done for defect models in behavioral designs.

II. Approach

We have developed a behavioral validation defect model which is based on previous research in *mutation analysis* for software test-

ing [6]. Mutation analysis has been used to generate hardware manufacture tests [4], but it has not been previously used for hardware design validation. In mutation analysis terminology, a *mutant* is a version of a software program 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, and therefore defines the behavioral validation defect model. Since behavioral hardware descriptions share many features in common with procedural software programs, we have used a subset of the software mutation operations presented in [6] as the core of our initial defect model. We have slightly modified these mutation operators to match the syntax of behavioral VHDL. The mutation operations which we are currently using are described below, and Table I contains examples of each operation applied to a line of VHDL code.

- **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.
- **Logical Operator Replacement (LOR)**
Each occurrence of one of the logical operators (and, or, xor) is replaced by each of the other operators; in addition, each is replaced by FALSEOP, TRUEOP, LEFTOP and RIGHTOP.
- **Relational Operator Replacement (ROR)**
Each occurrence of one of the relational operators (<, >, <=. >=, =, /=) 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 program unit is replaced by every compatible variable in the program.

Types	Good Machine	Mutant
AOR	$x := x - y$	$x := x + y$
LOR	$x \ and \ y$	$x \ or \ y$
ROR	$rst = true$	$rst /= true$
VR	$y := h$	$y := x$

TABLE I
EXAMPLES OF MUTANTS

We have developed a software tool which parses a behavioral VHDL description and automatically generates all mutants. By simulating the mutants and the original code with a set of test patterns, we can determine the coverage for each class of mutants. By comparing the mutant coverage to the coverage provided by a functional fault coverage metric, we can evaluate the accuracy of the fault coverage metric. We can also diagnose the weaknesses in an existing metric by identifying the class of mutants in which the accuracy is most degraded.

III. PRELIMINARY RESULTS

As an initial exploration into this approach, we have generated all mutants of the GCD VHDL benchmark example. The total number of mutants for the GCD example is 81, and the breakdown according to operator type is shown in Table II. The mutant operator $VR(c)$ includes VR mutants in conditional statements, while $VR(s)$ includes VR mutants in non-conditional statements. The original GCD code and all mutant versions of the code were simulated with a set of pseudo-random test patterns to determine the coverage of each class of mutants. The statement coverage of the original circuit was also computed, and all coverage results are shown in Figure 1.

Although the GCD is a single example, its validation results reveal a weakness in the statement coverage metric. Notice that the accuracy of the statement coverage metric is worst for the LOR mutants. The only line in

Type	Number of Mutants
AOR	3
VR(c)	12
VR(s)	46
ROR	15
LOR	5

TABLE II
GCD, NUMBER OF MUTANTS OF EACH TYPE

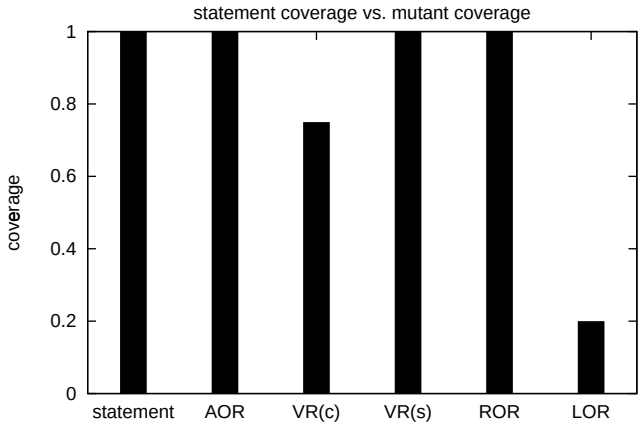

Fig. 1. GCD, statement coverage vs. mutation coverage

the GCD description which involves a logic operation is the following conditional statement:

```
if (x /= 0) and (y /= 0) then
```

The *else* clause of this conditional does not contain any statements. As a result, the fact that the else clause is never executed is not reflected in the statement coverage metric. A solution to this problem would be to use a branch coverage metric in addition to statement coverage.

IV. FUTURE WORK

We intend to use our mutation analysis tool to evaluate statement coverage, branch coverage, and other functional fault models. We will use a large number of high-level synthesis benchmarks to allow our results to be generalized more broadly. Because the mutation operations are local, we expect that the number of mutants will increase linearly with the number of lines in the behavioral description. This will allow us to use much larger benchmark examples with relatively low computational effort.

REFERENCES

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

[2] J. T. deSousa, F. M. Goncalves, J. P. Teixeira, C. Marzocca, F. Corsi, and T. W. Williams. Defect Level Evaluation in an IC Design Environment. *IEEE Transactions on Computer-Aided Design*, 15(10):1286–1293, October 1996.

[3] F. Fallah, P. Ashar, and S. Devadas. Simulation Vector Generation from HDL Descriptions for Observability Enhanced-Statement Coverage. In *Proceedings of the* 36^{th} *Design Automation Conference*, pages 666–671, June 1999.

[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] S. Kang and S. A. Szygenda. Design Validation: Comparing Theoretical and Empirical Results of Design Error Modeling. *IEEE Design & Test of Computers*, 11(1):18–26, Spring 1994.

[6] 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.