MICRO-OPERATION PERTURBATIONS IN CHIP LEVEL FAULT MODELING

Chien-Hung Chao, F. Gail Gray

Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060

ABSTRACT

The goal of this paper is to determine the best micro-operation perturbation for modeling faults at the chip level. The measure used is the gate level stuck-at fault coverage achieved by the tests derived to cover the micro-operation perturbation faults. For small combinational circuits, it is shown that perturbing the elements into the logic dual is a good choice. For large combinational circuits, it is shown that there is very little variation in the gate level coverage achieved by the various micro-operation faults. In this case, if coverage is to be improved, the micro-operation perturbation method must be augmented by other techniques.

INTRODUCTION

Because of the increasing complexity in digital VLSI devices, gate level testing becomes very difficult and expensive. Also, from the viewpoint of the user, the limited information available to them frequently makes gate level testing impractical. This is why functional level (or chip level) testing is becoming more important.

Levendel and Menon [1] proposed a test generation algorithm using hardware description languages (HDLs) as description media. An extension of the D-algorithm was employed to generate tests from HDL constructs (e.g. IF-THEN-ELSE, CASE). Armstrong and Gupta [2, 7, 10] developed different HDL-based fault models (i.e., micro-operation faults and control faults). A chip level test generation algorithm was developed by Barclay [3,4]. The algorithm input VHDL (VHSIC Hardware Description Language) modeling programs and generated test vectors by using artificial intelligence techniques.

In this paper we try to determine the type of micro-operation perturbation in high level models that will provide the best gate level stuck-at fault coverage.

CHIP LEVEL FAULT MODEL

The chip level HDL model is composed of sequences of micro-operations and control constructs, such as IF-THEN-ELSE and CASE. Therefore, two groups of faults are defined [2]: the micro-operation faults and the control faults.

The control faults are modeled by perturbing the HDL control constructs. For example an IF-THEN-ELSE statement may fail to STUCK-AT-THEN or STUCK-AT-ELSE, in which the group of micro-operations under the THEN (ELSE) is executed regardless of the value of the control expression. A CASE statement may fail if no operation occurs for some particular case, i.e., when the faulty clause is chosen by the control expression, it does not execute.

The research described in this paper was supported in part by the National Science Foundation, IBM, Control Data Cooperation, MCC, and Virginia's Center for Innovative Technology

A micro-operation is a logic or arithmetic operation in a data assignment statement of a HDL description. A faulty micro-operation corresponds to a perturbed data assignment statement which results in the wrong operation being performed. A micro-operation can be perturbed in many ways. For instance, an AND operation can fail to the OR operation or to the EXCLUSIVE-OR (EXOR) operation. In this paper, we grade a micro-operation fault model by applying the test set that detects the perturbed micro-operation to a gate level model using the HILO simulation program. The resulting gate level stuck-at fault coverage is used as a measure of effectiveness for the micro-operation fault model.

MICRO-OPERATION PERTURBATIONS IN SMALL COMBINATIONAL CIRCUITS

For small combinational circuits, a hardware description language model can be derived from the logic diagram.

EXAMPLE 1:

For example, consider the circuit shown in Fig. 1. The circuit can be modeled by the following substitution statement.

G = (NOT A) OR (NOT B) OR (C AND D)

This statement contains five micro-operations (two NOT operations, two OR operations, and one AND operation). We can then consider various perturbations of the operations in the HDL description, derive tests for each perturbation, and evaluate the gate level stuck-at fault coverage of each test set.

To evaluate the gate level stuck-at fault coverage, it is convenient to know the coverage obtained by each input combination. This was obtained either by using the standard D-algorithm or, in some cases, by using the HILO simulator. The coverages for the example circuit of Fig. 1 is shown in Table 1. Each row of the table shows the gate level stuck-at fault coverage obtained by one input combination to the circuit. For example, the row for ABCD = 0000 shows that input combination 0000 detects line 5 stuck at 0 and line 7 stuck at 0. Therefore, input combination ABCD=0000 detects 2 of the 14 faults, or about 14% of the total number of stuck-at faults. The percentage coverage for each combination is given in the first column of the table. Observe that some combinations are better than others with the best being input combinations 1101 and 1110 (42%) and the worst being input combinations 0011, 0111, and 1011 (only 7%). Micro-operation perturbations that are detected by input combinations 1101 and 1110 will in some sense be better than those that are detected only by combinations 0011, 0111, and 1011. We will look at many circuits to see if we can predict which choices might be better.

Many types of micro-operation perturbations were considered. Table 2 shows the set of perturbations applied to the example in Fig. 1. There are five multiple perturbations, three single perturbations, and three-stuck at faults for comparison. The faulty functions generated by the micro-operation perturbations are shown in the third column. The average coverage for each perturbation represents the average stuck-at fault coverage for each of the input combinations that detects the perturbation. Table 3 shows the truth tables for the good function (G) and all of the faulty functions

CH2540-3/88/0000/0579$01.00 © 1988 IEEE

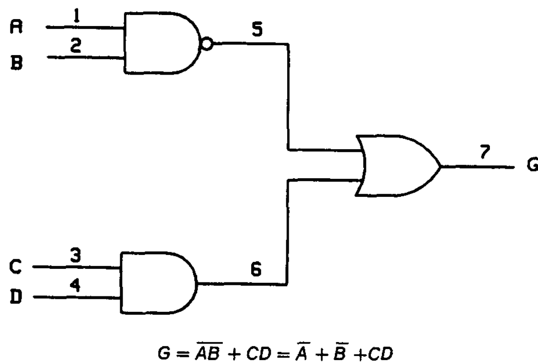
(G1 through G11) from column three of Table 2. For example, from Table 3 it is determined that input combinations 1101, and 1110 are the only input combinations that will detect perturbation G5. Therefore, the average coverage for these two combinations is 42%.

From similar experiments performed on several circuits, we draw the following conclusions.

1. Either AND -> OR or OR -> AND perturbations provide very high average coverage in all of the examples for both single and multiple perturbations. That is, either faults 4 or 5 are among the best for multiple perturbations, and either faults 7 or 8 are among the best for single perturbations.

2. Taking the dual of a function (AND -> OR and OR -> AND at the same time) may not result in high coverage. This indicates that taking the dual of a function may not work out well in general.

3. Different implementations of a function produce the same result. The best micro-operation perturbation appears to be independent of implementation.



$$G = \overline{AB} + CD = \bar{A} + \bar{B} + CD$$

Fig. 1. Circuit Diagram for Example 1

Table 1. Stuck-at Fault Detection Table

COV %	TEST	ABCD	1/0	1/1	2/0	2/1	3/0	3/1	4/0	4/1	5/0	5/1	6/0	6/1	7/0	7/1
14	0	0000									x				x	
14	1	0001									x				x	
14	2	0010									x				x	
7	3	0011													x	
21	4	0100		x							x				x	
21	5	0101		x							x				x	
21	6	0110		x							x				x	
7	7	0111													x	
21	8	1000				x					x				x	
21	9	1001				x					x				x	
21	10	1010				x					x				x	
7	11	1011													x	
35	12	1100	x		x							x		x		x
42	13	1101	x		x			x				x		x		x
42	14	1110	x		x					x		x		x		x
28	15	1111					x		x				x		x	

Table 2. Experimental Results for Example 1

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COV.
multiple perturbations :			
1	take dual function	$G1 = \bar{A}\bar{B}(C + D)$	18.2
2	take complement of G	$G2 = AB(\bar{C} + \bar{D})$	21.0
3	take complement of inputs	$G3 = A + B + \overline{CD}$	25.7
4	OR -> AND only	$G4 = \overline{AB}CD$	17.5
5	AND -> OR only	$G5 = \bar{A} + \bar{B} + C + D$	42.0
single perturbation :			
6	OR -> AND	$G6 = \bar{A}\bar{B} + CD$	21.0
7	OR -> AND	$G7 = \bar{A} + \bar{B}CD$	22.8
8	AND -> OR	$G8 = \bar{A} + \bar{B} + C + D$	42.0
some stuck-at faults for comparison			
9	NAND s-a-1	G9 = 1	39.7
10	AND s-a-0	$G10 = \overline{AB}$	28.0
11	OR s-a-1	G11 = 1	39.7

Table 3. Truth Tables for Micro-Operation Perturbations

COV %	TEST	ABCD	G 1	G 2	G 3	G 4	G 5	G 6	G 7	G 8	G 9	G 10	G 11
14	0	0000	1	0	1	1	0	1	1	1	1	1	1
14	1	0001	1	1	1	0	0	1	1	1	1	1	1
14	2	0010	1	1	1	0	0	1	1	1	1	1	1
7	3	0011	1	1	1	0	1	1	1	1	1	1	1
21	4	0100	1	0	1	1	0	1	0	1	1	1	1
21	5	0101	1	0	1	1	0	1	0	1	1	1	1
21	6	0110	1	0	1	1	0	1	0	1	1	1	1
7	7	0111	1	0	1	1	0	1	1	1	1	1	1
21	8	1000	1	0	1	1	0	1	0	0	1	1	1
21	9	1001	1	0	1	1	0	1	0	0	1	1	1
21	10	1010	1	0	1	1	0	1	0	0	1	1	1
7	11	1011	1	0	1	1	0	1	1	1	1	1	1
35	12	1100	0	0	1	1	0	0	0	0	1	0	1
42	13	1101	0	0	1	1	0	1	0	0	1	1	0	1
42	14	1110	0	0	1	1	0	1	0	0	1	1	0	1
28	15	1111	1	0	0	1	0	1	1	0	1	1	0	1

MICO-OPERATION PERTURBATIONS IN LARGE COMBINATIONAL CIRCUITS

From the previous section on small combinational circuits, we got some idea about what kind of micro-operation fault model may be acceptable, e.g., taking the logic dual of a small operator. The next step is to see whether those observations also hold for large combinational circuits.

When we deal with chip level descriptions of large circuits, usually there are a very large number of inputs that can detect each micro-operation fault. This makes it very difficult to compute the exact expected coverage. A statistical method [6] was used to estimate the average coverage for a test set by randomly sampling the set and calculating the average coverage for the random sample. The confidence interval for the average was computed. If the confidence interval was too broad, more samples were added until the confidence interval was acceptable (+ or - 2%). The 95% confidence interval for fault coverage based on the student's T distribution is:

$$\bar{c} - t_{0.05}(df) \times s/\sqrt{n} \leq u \leq \bar{c} + t_{0.05}(df) \times s/\sqrt{n}$$

Where

u is the estimated coverage for the test set,

$\bar{c}$ is the arithmetic mean of coverages for a sample of size n,

df is the degree of freedom (df = n - 1),

$t_{0.05}(df)$ is the 95% confidence student's t quantity, which is a function of degree of freedom,

s is the sample standard deviation which equals,

$$\sqrt{\sum_{i=1}^{n}(c_i - \bar{c})^2/(n-1)}$$

n is the sample size,

d is the interval distance, which is half of the length of the interval,

$$t_{0.05}(df) \times s/\sqrt{n}$$

The maximum interval distance (d) is set as 0.02 (2%) for good resolution.

Although VHDL provides a more powerful set of constructs for modeling at the chip level [7,8], we adopted GSP2 [9] as the working hardware description language because VHDL simulation is not efficient at this time.

The procedure for calculating the 95% confidence coverage interval using a random sampling technique is summarized below:

1. Write a chip level description for the circuit under test and verify it through simulation.
2. Create a faulty module by perturbing some micro-operation in the good model, e.g., AND failed to OR.
3. Find the test set which can detect the micro-operation fault. This is done by applying input combinations exhaustively to both good and faulty modules. If an input combination causes different outputs, it is a test for the micro-operation fault.
4. Randomly pick one test from the test set.
5. Input the test into the HILO simulator to get the gate level stuck-at fault coverage for the test.
6. If an acceptable 95% confidence interval has been achieved, stop the procedure; otherwise, go to 4.

By comparing 95% confidence intervals of various micro-operation fault models, we can decide which one is the best choice.

The random sampling technique is coded and the procedure of calculating the 95% confidence intervals is automated by a UNIX Bourne Shell program. An example is given below :

EXAMPLE 2 :

The chip level description for the 4-bit carry-look-ahead adder (SN7483a) contains the following micro-operation :

```
int_result1 := @ADD(@CONCAT(extended_bit,a),
                    @CONCAT(extended_bit,b));
```

Where int_result1 is a temporary 5-bit vector ranging from bit 0(LSB) to bit 4(MSB), and a and b are 4-bit input vectors. The extended_bit which has value 0 is concatenated with a and b to calculate the carry_out bit.

Two classes of faults are considered, micro-operation substitution faults and micro-operation stuck-at faults. The micro-operation substitution fault is to substitute the good operator with a faulty one, e.g., ADD failed to SUB. The micro-operation stuck-at fault is to force the outcome of the operator to stuck at one or zero. An example is given below for each fault model class.

```
micro-operation substitution fault :
   int_result1 := @SUB(@CONCAT(extended_bit,a),
                       @CONCAT(extended_bit,b));
micro-operation stuck-at fault :
   int_result1 := @ADD(@CONCAT(extended_bit,a),
                       @CONCAT(extended_bit,b));
   int_result1 := @CONCAT(int_result[1|4], #B0);
```

Fig. 2 shows how to obtain a test set for a given fault, where exhaustive input combinations are fed into both good and faulty modules. The COMP module compares the outputs from the good and the faulty modules. Whenever an input combination causes different outputs, it is a test for the micro-operation fault under evaluation. The GSP2 output LOG FILE is handled by the EXTRACT program, which is a Shell script, to extract the test set TEST. In this way, a test set can be obtained.

Next, the test set is input to the system shown in Fig. 3. WRITE.P inputs the test set from the TEST module, the initial random seed and the initial sample size N, which is zero, from the NSEED module. It then randomly picks a test from the test set and writes the waveform file for it. The waveform file is input to the HILO fault simulator. The coverage for that test is written into COV file. The INT.P module accepts the coverages from previous random samplings and determines whether the 95% confidence interval has been achieved. If the interval has been achieved, the RESULT file is created and the whole process stops; otherwise, another test is picked and the procedure is repeated again. The DEMO module stores information of each round and TTABLE keeps the 95% confidence t values. The SCRIPT, which is a UNIX Bourne Shell program, is the master of the system. It automates the random sampling technique.

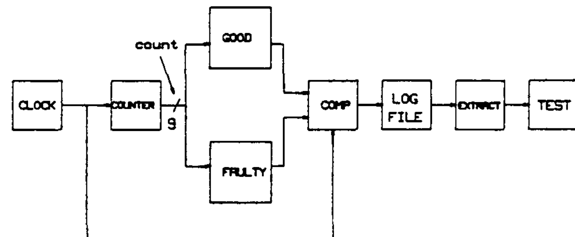


Fig. 2. Using GSP2 to Obtain the Test Set

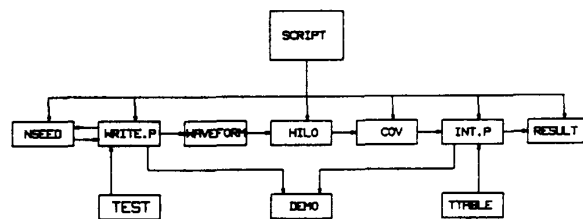


Fig. 3. UNIX Script to Automate the Experimental Procedure

The experimental results are summarized in Table 4. The table indicates that all of the micro-operation faults result in about the same average fault coverage. We also experimented with SN74181 ALU, which is a good circuit to try since it exercises various logic functions and arithmetic functions. The results also showed no best micro-operation fault model. The reason for this is that usually there are many tests which can detect each micro-operation fault. Consequently, the test sets for various micro-operation fault models often overlap one another and that, in turn, results in similar coverage intervals.

CONCLUSIONS

An automatic procedure, which employs a statistical random sampling technique, is presented to search for best micro-operation fault model in chip level testing using HDL. For small combinational circuits, micro-operations perturbing to the logic dual seems to provide consistently high coverage. Although the chip level experiments revealed no best micro-operation fault model for large combinational circuits, the logic dual fault model is still a good choice because it is as good as other micro-operation fault models for large circuits and it is better for small circuits.

REFERENCES

[1] Y. Levendel and P. Mellon, "Test generation algorithms for computer hardware description languages", *IEEE Trans. Computers, pp. 577-588, July 1982.*

[2] J. R. Armstrong, "Chip-Level Modeling with HDLs", *IEEE Design & Test of Computers, pp. 8-18, Feb. 1988.*

[3] D. S. Barclay, "An Automatic Test Generation Algorithm for Chip-Level Circuit Descriptions", *Master's Thesis, Virginia Polytechnic Institute and State University, January 1986.*

[4] D. S. Barclay and J. R. Armstrong, "A Heuristic Chip-Level Test Generation Algorithm", *23rd Design Automation Conference, pp. 257-262, June 1986.*

[5] J. Roth, "Diagnosis of automata failures : a calculus and a method", *IBM J. of R&D, July 1966.*

[6] G. W. Snedecor and W. G. Cochran, Statistical Methods, *The Iowa State University Press, 1980.*

[7] J. R. Armstrong, Chip Level Modeling With VHDL, to be published.

[8] D. Han, "Optimal Constructs for Chip Level Modeling", *Master's Thesis, Virginia Polytechnic Institute and State University, August 1986.*

[9] J. M. Kerr and C. H. Cho, "GSP2 Hardware Description and Modeling Language Reference Manual", *E. E. Dept., Virginia Polytechnic Institute and State University, Feb. 1986.*

[10] A. K. Gupta, "Functional Fault Modeling and Test Vector Development for VLSI Systems", *Master's Thesis, Virginia Polytechnic Institute and State University, March 1985.*

Table 4. Experimental Results for SN7483a

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
micro-operation substitution fault :				
1	ADD -> SUB	481	12	[0.349, 0.389]
2	ADD -> AND	510	12	[0.357, 0.394]
3	ADD -> OR	350	9	[0.348, 0.386]
4	ADD -> EXOR	350	9	[0.365, 0.401]
int_result1 stuck-at faults :				
5	ADD -> 11111	512	3	[0.357, 0.386]
6	ADD -> 00000	510	7	[0.348, 0.388]
7	ADD -> 10101	492	6	[0.362, 0.398]
8	ADD -> 01010	490	10	[0.368, 0.406]
9	bit 0 s-a-0	256	9	[0.377, 0.413]
10	bit 0 s-a-1	376	9	[0.349, 0.387]
11	bit 1 s-a-0	257	5	[0.366, 0.395]
12	bit 1 s-a-1	255	5	[0.346, 0.385]
13	bit 2 s-a-0	256	6	[0.355, 0.394]
14	bit 2 s-a-1	256	4	[0.360, 0.392]
15	bit 3 s-a-0	258	6	[0.372, 0.410]
16	bit 3 s-a-1	255	5	[0.352, 0.380]
17	bit 4 s-a-0	242	10	[0.343, 0.380]
18	bit 4 s-a-1	271	9	[0.379, 0.416]