MICRO-OPERATION PERTURBATIONS IN CHIP LEVEL FAULT MODELING

by

Chien-Hung Chao

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. F. Gail Gray, Chairman

Dr. James R. Armstrong

Dr. Hugh F. Van Landingham

February, 1988

Blacksburg, Virginia

MICRO-OPERATION PERTURBATIONS IN CHIP LEVEL FAULT MODELING

by

Chien-Hung Chao

Dr. F. Gail Gray, Chairman

Electrical Engineering

(ABSTRACT)

In chip level testing using hardware description language approach, a difficult question to answer is : What is the best micro-operation perturbation for modeling fault at the chip level ? In this thesis, an automatic evaluation system is developed to determine the best micro-operation perturbation. 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.

Acknowledgements

I would like to express my deepest respect and thanks to my advisor, Dr. F. G. Gray, for his great guidance and encouragement toward this work. It has been an honor and a pleasure to work under him.

I also like to thank Dr. J. R. Armstrong and Dr. H. F. Van Landingham, my committee members, for their great efforts and help. Additional thanks are due to Dr. J. R. Armstrong. His support and guidance play an important role in this work.

Lastly, I dedicate my work to my parents and my wife, Jing-Wen, for their love and support.

Table of Contents

1.0 Introduction **1**

1.1 The Importance of Testing 1

1.2 Concept of Testing 1

1.3 Needs for Functional Level Testing 3

1.4 Chip Level Testing 4

1.5 Objectives of This Thesis 5

1.6 Outline of Contents 6

2.0 Literature Review **7**

2.1 D - algorithm 7

2.2 System Graph Method 8

2.3 State Transformation Graph Method 9

2.4 Functional Solution Language Method 10

2.5 The S-algorithm 10

2.6 Hardware Description Language Method 11

3.0 Evaluation Method, Experiments and Analysis for Small Circuits **12**

3.1 Evaluation Method for Small Combinational Circuits 13
3.2 Experiments for Small Combinational Circuits 15
3.2.1 Circuit 1 15
3.2.2 Circuit 2 22
3.2.3 Circuit 3 22
3.2.4 Circuit 4 22
3.2.5 Circuit 5 23
3.2.6 Circuit 6 23
3.2.7 Circuit 7 24
3.2.8 Circuit 8 24
3.2.9 Circuit 9 25
3.2.10 Circuit 10 25
3.2.11 Circuit 11 26
3.2.12 Circuit 12 26
3.2.13 Circuit 13 26
3.2.14 Circuit 14 27
3.2.15 Circuit 15 27
3.3 Analysis Summary for Small Combinational Circuits 57

4.0 Evaluation Method, Experiments and Analysis for Large Circuits 58
4.1 Evaluation Method for Large Combinational Circuits 59
4.2 Experiments for Large Combinational Circuits 62
4.2.1 Experiments with SN7483a Carry-Look-Ahead adder 64
4.2.2 Experiments with SN74181 ALU 76
4.3 Analysis Summary for Large Combinational Circuits 88
4.4 Suggestions for Future Work 89

5.0 Conclusions 90

Bibliography **91**

Appendix A. GSP2 Programs for SN7483A Adder Experiments **93**

A.1 The CONNECT Program 94

A.2 The CLOCK Module Program 95

A.3 The COUNTER Module Program 96

A.4 The Good Module Program for the SN7483a Adder 97

A.5 The Faulty Module Program for the SN7483a Adder 98

A.6 The COMP Module Program 99

A.7 The GSP2 Command Language File 100

Appendix B. GSP2 Programs for SN74181 ALU Experiments **101**

B.1 The CONNECT Program 102

B.2 The CLOCK Module Program 104

B.3 The COUNTER Module Program 105

B.4 The Good Module Program for the SN74181 ALU 106

B.5 The Faulty Module Program for the SN74181 ALU 110

B.6 The COMP Module Program 114

B.7 The GSP2 Command Language File 115

Appendix C. Programs for Calculating the Intervals **116**

C.1 The SCRIPT Module Program 117

C.2 The EXTRACT Program 119

C.3 The WRITE.P Module Program 120

C.4 The HILO.CMDS Program 123

C.5 The EX.CMDS Program 124

C.6 The INT.P Module Program 125

C.7 The TTABLE Module t Data 128

Vita 129

List of Illustrations

Figure 1. Circuit Diagram for Circuit 1 16
Figure 2. Circuit Diagram for Circuit 2 28
Figure 3. Circuit Diagram for Circuit 3 30
Figure 4. Circuit Diagram for Circuit 4 32
Figure 5. Circuit Diagram for Circuit 5 34
Figure 6. Circuit Diagram for Circuit 6 36
Figure 7. Circuit Diagram for Circuit 7 38
Figure 8. Circuit Diagram for Circuit 8 40
Figure 9. Circuit Diagram for Circuit 9 43
Figure 10. Circuit Diagram for Circuit 10 45
Figure 11. Circuit Diagram for Circuit 11 47
Figure 12. Circuit Diagram for Circuit 12 49
Figure 13. Circuit Diagram for Circuit 13 51
Figure 14. Circuit Diagram for Circuit 14 53
Figure 15. Circuit Diagram for Circuit 15 55
Figure 16. Using GSP2 to Obtain the Test Set 66
Figure 17. HP-UX Script to Automate the Experimental Procedure 68

List of Tables

Table 1. Stuck-at Fault Detection Table 19
Table 2. Experimental Results for Circuit 1 20
Table 3. Truth Tables for Micro-Operation Perturbations 21
Table 4. Experimental Results for Circuit 2 29
Table 5. Experimental Results for Circuit 3 31
Table 6. Experimental Results for Circuit 4 33
Table 7. Experimental Results for Circuit 5 35
Table 8. Experimental Results for Circuit 6 37
Table 9. Experimental Results for Circuit 7 39
Table 10. Experimental Results for Circuit 8 - C part 41
Table 11. Experimental Results for Circuit 8 - S part 42
Table 12. Experimental Results for Circuit 9 44
Table 13. Experimental Results for Circuit 10 46
Table 14. Experimental Results for Circuit 11 48
Table 15. Experimental Results for Circuit 12 50
Table 16. Experimental Results for Circuit 13 52
Table 17. Experimental Results for Circuit 14 54
Table 18. Experimental Results for Circuit 15 56
Table 19. A Part Results for SN7483a First ADD Micro-operation 71
Table 20. A Part Results for SN7483a Second ADD Micro-operation 72
Table 21. B Part Results for SN7483A 73

Table 22. C Part Results for SN7483A 74
Table 23. D Part Results for SN7483A 75
Table 24. A Part Results for SN74181 ALU 78
Table 25. A Part Results for SN74181 ALU (Continued) 79
Table 26. A Part Results for SN74181 ALU (Continued) 80
Table 27. A Part Results for SN74181 ALU (Continued) 81
Table 28. A Part Results for SN74181 ALU (Continued) 82
Table 29. A Part Results for SN74181 ALU (Continued) 83
Table 30. A Part Results for SN74181 ALU (Continued) 84
Table 31. B Part Results for SN74181 ALU 85
Table 32. C Part Results for SN74181 ALU 86
Table 33. D Part Results for SN74181 ALU 87

1.0 Introduction

1.1 The Importance of Testing

All integrated chips should be tested before they are shipped out to users. This is because flaws in the masks, defects in original silicon wafer, short circuits in metal, diffusion or polysilicon inter-connection, etc., all cause defective chips. Of the large number of integrated chips fabricated on a single silicon wafer, only a fraction will be completely functional. Also, from the viewpoint of users, integrated chips must be tested before being adopted in an individual application. As a result, testing is a very important process in terms of quality control of chip manufacture.

1.2 Concept of Testing

Testing here means fault detection, that is, the discovery of something wrong in a chip. This is done by applying a sequence of test inputs to the chip-under-test using automatic test equipment

(ATE) comparing the resulting outputs with the reference outputs [1,3,5]. Therefore, it is an important issue in testing to generate good tests that can detect all - or, more likely, almost all - physical faults in a chip.

Test generation is carried out with the help of simulation. A digital system can be modeled at different levels, i.e., processor memory switch (PMS), chip, register, gate, circuit, and silicon [2,10]. Once the level of abstraction is decided, the digital system under test is modeled with a well-defined description medium such as System Graph [24], VHDL (VHSIC Hardware Description Language) [14,15], then, a fault model is set up to map physical faults to abstract or functional faults. A test generation algorithm is then developed to generate tests for all the abstract faults in the modeled digital system. If experiment or analysis can show that these tests will detect all or almost all physical faults in the digital system, the fault model is considered sufficiently accurate for test generation.

Fault modeling deals with the systematic and precise mapping from physical faults to abstract faults suitable for simulation and test generation. Different fault models are necessary for each level of abstraction at which the circuits of interest are modeled. A good fault model is the base for a successful automatic test generation algorithm. What constitutes a good fault model? First, it should match the abstraction level at which it is to be used. Second, the complexity and the number of the abstract faults should not result in excessive computation time. Finally, the faulty behavior of the physical defects should be reflected with sufficient accuracy by the abstract fault model in order to generate good tests [3,4].

Once a good fault model is set up, the tests generated by another manual process or by an automatic test pattern generation (ATPG) system can be evaluated by a fault simulator through fault simulation. A test T usually consists of a sequence of test patterns which spread over some time span, $T = \{t1,t2,...,tk\}$. The fault simulator first creates a fault list for the modeled circuit. If the circuit has n modeled faults, the fault simulator generates $n + 1$ models, i.e. the fault-free model and n faulty models each containing exactly one modeled fault. Fault simulation is the process of applying every test pattern to the fault-free model, and to each of the n faulty models. After fault

simulation is finished, test coverage is calculated, which is the ratio of the number of faults detected (m) to the total number of faults (n), as a merit measure of test T.

In Summary, a complete testing process usually includes the following steps:

1. Decide the level of abstraction, for example, chip level.

2. Describe the circuit under test in terms of a well defined description medium such as State Transformation Graph [25], HILO [30].

3. Set up a good fault model (or models).

4. Develop a test generation algorithm to generate tests, each test may consist of a sequence of test patterns.

5. Evaluate the tests through fault simulation.

1.3 Needs for Functional Level Testing

Over the past years, the gate level stuck-at fault model along with an automatic test generation program implementing the D-algorithm [6] and its extensions has been employed for logic testing [7].

As time passes and VLSI technology evolves every day, the number of devices per chip increase dramatically. For example, today Motorola's MC68020 32-bit microprocessor contains 200,000 transistors on a 350x375 mils chip. The increasing complexity in VLSI chips makes gate level testing very difficult and expensive [5]. Also, from the user's point of view, the limited information available to them (mostly functional) frequently makes gate level testing impractical. This

is why functional level testing is becoming more important [23]. Functional level testing uses a representation of digital system higher than gate level, for instance, register level or chip level.

1.4 Chip Level Testing

Among various functional testing methods is chip level testing using hardware description languages (HDLs) as description media. In Levendel and Menon [8], an extension of the D-algorithm was employed to generate tests from HDL constructs (e.g. IF-THEN-ELSE, CASE).

Armstrong and Gupta [2,9,10,11] proposed different HDL-based fault models at the chip level. A VLSI circuit is modeled as a single entity, not as a hierarchy of low-level primitives, in the chip level modeling. 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 [2,10] : 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(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.

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.

The fault models are graded by applying the test set that detects the chip level faults in a chip level model to a gate level model of the same circuit, because there are no ATE available and gate level stuck-at fault simulation is widely employed in industry.

Gupta manually generated tests for a 3700-gate signal processing chip described in GSP, an assembly language-style HDL [16,17]. The chip level faults gave 88.6% coverage of gate level stuck-at faults [11,18]. Additional experiments were done to verify the fault model [19]. Eleven small/medium-size circuits were modeled in GSP2, a block- structured language [20]. Tests were derived manually from the chip level models using the chip level fault models. These tests were run on gate level models of the same circuits, yielding an average of 92.4%. Therefore, the chip level fault models appear valid.

Based on the micro-operation faults and the control faults, Barclay [12,13] developed a chip level test generation algorithm. The algorithm input VHDL (VHSIC Hardware Description Language) [14,15] modeling programs and generate test vectors by using the artificial intelligence concept of goal tree solving. An AND-OR goal tree is an artificial intelligence data structure used to solve problems by breaking a large problem into smaller and smaller pieces until each piece is small enough to be solved directly. The algorithm first pick an item in the source VHDL program to fault and determines the basic test requirements in the form of basic test goals. This process is similar to deriving primitive D-cubes in the D-algorithm. Second, the algorithm tries to solve the basic test goals, i.e., forward drive the effect of the fault from its original location to the modeled circuit output and backward drive the basic test goals to establish the necessary inputs. While in this process, a goal tree forms and goals are solved by breaking them into subgoals which are then solved. The process terminates with a set of input conditions (test patterns) required to propagate the effects of the fault to a system output.

1.5 Objectives of This Thesis

This thesis tries to determine the type of micro-operation perturbation needed in Barclay's test generation algorithm that will provide the best gate level stuck-at fault coverage. The set of tests that detects a micro-operation fault is applied to a gate level model of the same circuit for fault simu-

lation. The resulting gate level stuck-at fault coverage is use as a measure of effectiveness for the micro-operation fault model.

1.6 Outline of Contents

This chapter first depicts the importance of testing and the concept of testing. Then, needs for functional level testing is pointed out because of the increasing complexity in VLSI chips. Finally, chip level testing with HDLs is introduced and the goal of this thesis is presented.

Chapter 2, "Literature Review", is a review of previous testing methods, emphasis is put on functional testing.

Chapter 3, "Evaluation Method, Experiments and Analysis for Small Circuits", describes the micro-operation perturbation evaluation method for small combinational circuits. This chapter also uses an example to illustrate the experimental procedures. The experimental results of other sample circuits are presented and the analysis for the results is given.

Chapter 4, "Evaluation Method, Experiments and Analysis for Large Circuits", depicts the micro-operation perturbation evaluation method for large combinational circuits, demonstrates the experimental procedures with an example and analyzes the experimental results. Some suggestions for future work are also given here.

Chapter 5, "Conclusions", gives conclusions about the perturbation evaluation methods.

Appendix A, "GSP2 Programs for SN7483a Adder Experiments", presents GSP2 programs used in the experiments of SN7483a adder.

Appendix B, "GSP2 Programs for SN74181 ALU Experiments", presents GSP2 programs used in the experiments of SN74181 ALU.

Appendix C, "Programs for Calculating the Coverage Intervals", lists Pascal and HP-UX Shell programs used in the experiments of large circuits.

2.0 Literature Review

Because of the increasing complexity in digital VLSI devices, gate level testing becomes very time consuming and difficult. This creates a major problem in modern VLSI technology. There are needs for both manufacturers and users to find reliable, low-cost, comprehensive testing techniques. Functional testing is receiving more and more attention during recent years. In this chapter, gate level D-algorithm is first surveyed and then several functional testing techniques are described in the succeeding sections.

2.1 D - algorithm

Roth's D-algorithm is the first algorithmic method for generating tests at the gate level [6]. The symbol D at a node means a good value of 1 is failed to 0 and $\overline{D}$ means a good value of 0 is failed to 1. For each gate in the circuit, primitive cubes are first calculated, which is basically a compact version of the gate's truth table. Singular cover is the set of all primitive cubes for all gates in the circuit network. Then, primitive cubes with different output values of each gate are intersected to derive the propagation D-cubes, which propagate faults on gate inputs to gate outputs. Single faults

are assumed. The primitive D-cubes of a fault specifies the minimal conditions required to force the faulty gate output to be different from the correct gate output. After a primitive D-cube of a fault is derived, D-drive is performed to sensitize all possible paths from the faulty gate to a primary output of the circuit. The final step is the line justification, which is performed to develop a consistent set of primary input values, or test.

A lot of work has been done on D-algorithm and its extensions [3,21,22].

2.2 *System Graph Method*

Thatte and Abraham [24] proposed a functional test generation algorithm for testing microprocessors in a user environment. A microprocessor is modeled at the register transfer level by a system graph called S-graph. An S-graph is a directed graph in which each node represents a register and each edge represents the execution of instructions or subinstructions. A subinstruction is defined as one of several register transfer level operations needed to complete an instruction.

The method considered the following microprocessor functions: register decoding, instruction decoding and control, data storage, data transfer, and data manipulation. An individual functional fault model is derived from each of these primary functions. In the register decoding fault model, none or multiple registers are accessed when a fault occurs. For the instruction decoding and control function, faulty behaviors considered are no instruction being executed, a wrong instruction being executed or an extra instruction being executed. Stuck-at faults and bridging faults are considered in data storage and data transfer faults. No fault model is given for the data manipulation function.

Individual fault testing procedures are proposed to test these functional faults (except data manipulation function). The algorithm is used to test a real 8-bit microprocessor. The resulting single stuck-at fault coverage was about 90%.

2.3 *State Transformation Graph Method*

A register transfer level testing methodology was proposed by Lai [25] based on a graph language called State Transformation Graph (STG). Unlike the System Graph method which applies to microprocessors, STG was designed to represent the general digital system under test. An STG is a directed graph in which links represent data or control paths while nodes represent data transformation operators. The STG is data-driven in the sense that a node is "fired" only if all of its input tokens are available.

Two fault models are actually considered : single-path model and single-node model. No faulty behaviors are specified. Once a fault model is chosen, a set of graph primitives are selected from STG for test generation. For each member of the set, the"functional analyzer" generates a parameterized test by parameter introduction, backward propagation, forward propagation, and justification. A parameterized test consists of a pair of symbolic assertions. The first assertion states the condition that must exist before the test cycle. The second assertion states the expected changes in the machine state after the test cycle. At the end of the above process, a set of parameterized tests is created. Then, the "test case synthesizer" is called to substitute the formal parameters by bit patterns stored in a test patterns database. The database stores test patterns for each graph primitive developed through lower-level, more implementation-oriented fault models. Finally, the "test program synthesizer" is called to map these test vectors into test program segments.

The single-path model was used to generate test program segments for PDP-8. Fault simulations were performed for the test program segments against single bit stuck-at faults on the data paths. The test program segments performed better than the diagnostics programs provided by PDP-8 manufacturer (DEC) in the sense that better fault coverage was achieved using less number of instructions.

2.4 *Functional Solution Language Method*

In [26], Breuer and Friedman proposed an automatic test generation algorithm for complex sequential circuits containing functional level primitives such as counters or shift registers. A test generation algorithm similar to the D-algorithm is employed to test each test point within the circuit. The Functional Solution Language (FSL) was designed, based on the functions of the primitive functional elements, to specify solutions to the D-drive, line justification and implication problems.

The circuit under test is first described as a mixture of basic logic gate, flip-flop and functional primitives. Then the functional behaviors of each functional primitive are mapped into a set of algorithms. After a stuck-at fault is inserted, the functional solution language is used to derive solution sequences during the D-drive, line justification and implication processes. The counter and the shift register were used to demonstrate the method.

2.5 *The S-algorithm*

Su and Lin [27, 28] proposed the S-algorithm, where S stands for symbolic, to generate test patterns for detecting functional faults at the register transfer level. The S-algorithm employs the symbolic execution technique to generate test patterns. Symbolic execution is like normal program execution except that symbolic values are used instead of actual variable values. Nine categories of functional faults are set up based on possible failure modes of various elements of the register transfer language description. For example, function decoding fault means the operation on source registers is faulted to an erroneous one.

A register transfer language (RTL) is defined to describe the circuit under test. The RTL description of the circuit is first partioned into a set of function submodules. Then, according to the

test order among the function submodules, a function submodule is picked and symbolic execution is performed for the fault-free function submodule and each functional fault injected in the function submodule. A test pattern is derived for each injected fault by comparing the symbolic results of the fault-injected and fault-free symbolic executions. The process is repeated for all the function submodules.

2.6 *Hardware Description Language Method*

An extension of the D-algorithm was proposed by Levendel and Menon [8] to generate tests from high level computer hardware description language (CHDL) models. The CHDL description of the circuit under test is comprised of a sequence of CHDL statements linked by control constructs such as IF-THEN-ELSE or CASE. The fault modes considered are input, output, and state variable stuck-at faults, control faults, and general function faults. The general function faults are user-specified faulty behaviors whose effects are known, but cannot be modeled by stuck variables or control faults.

Propagation D-cubes are derived for boolean switching expressions, and also for non-switching operations such as shifting, addition and counting. The IF and CASE control constructs are transformed to switching expressions to derive propagation D-cubes. The test generation procedure is similar to the D-algorithm, i.e., fault injection, D-drive, and line justification.

3.0 Evaluation Method, Experiments and Analysis for Small Circuits

The goal of this thesis is to determine the best micro-operation perturbation for modeling faults at the chip level. This chapter first describes the evaluation method for various micro-operation perturbations in small combinational circuits. For small combinational circuits, an output in a logic diagram is mapped to a boolean function and an HDL assignment statement. The average coverage for each micro-operation perturbation in the HDL assignment statement is used as the comparison metric for various micro-operation perturbations. After the evaluation method is described, an example is given to illustrate the experimental procedures for small combinational circuits. Experimental results and analysis of other sample circuits are also given. Finally, the analysis summary of the experimental results is presented.

3.1 *Evaluation Method for Small Combinational Circuits*

For small combinational circuits, a hardware description language model can be derived from the logic diagram. That is, each output in a logic diagram is first represented by a boolean equation and then the boolean equation is mapped to an HDL assignment statement containing one or more micro-operations. Note that the HDL statements do not necessarily reflect the actual structure of the circuit because many forms of a boolean equation can be derived by using De Morgan's laws. For each HDL statement (hence each good output function G), we can then consider various perturbation of the operations in the HDL description, derive tests for each perturbation and evaluate the average gate level stuck-at fault coverage of each test set.

To evaluate the average gate level stuck-at fault coverage of a test set, we have to know first the coverage obtained by each input combination. This was doned by using the standard D-algorithm or, in some cases, by using the HILO fault simulator[30]. The information is depicted in a table called "Stuck-at Fault Detection Table" for later use.

Many types of micro-operation perturbations were considered. They are summarized below:

1. Multiple perturbations : More than one micro-operations in a HDL statement (hence in a good output function G) are faulted.

   a. Take dual function : All micro-operations in G are faulted to their dual operations. For example, AND failed to OR and OR failed to AND.

   b. Take complement of G : The good output function G is faulted to its complement.

   c. Take complement of inputs : The inputs of G are faulted to their complements.

   d. OR -> AND only : Only OR operations in G are faulted to AND.

e. AND -> OR only : Only AND operations in G are faulted to OR.

2. Single perturbation : Only one micro-operation in G is faulted at one time.

   a. OR -> AND : One OR operation in G is faulted to AND.

   b. AND -> OR : One AND operation in G is faulted to OR.

3. Input stuck-at fault : Some input variable is forced to be stuck at constant 1 or 0.

A table called "Truth Tables for Micro-Operation Perturbations" shows the truth tables for the good function (G) and all of the perturbed functions. A set of tests can be obtained for each faulty function by comparing the truth table of the good function (G) and the truth table of the faulty function. If an input combination produces different outputs between the good function and the faulty one, it is a test for the faulty function and the corresponding micro-operation perturbations.

After a test set for a micro-operation perturbation is obtained, we enumerate the average coverage for the perturbation using the coverage information listed in the "Stuck-at Fault Detection Table" mentioned above. The average coverage for each perturbation represents the average stuck-at fault coverage for each of the input combinations that detects the perturbation. If the average coverage for some perturbation is higher than the average coverages for other perturbations, this perturbation is better than others because the probability for its individual corresponding test to detect more faults is higher than that of other perturbations. Therefore, the average coverage for each perturbation is used as an index of effectiveness for comparison.

3.2 *Experiments for Small Combinational Circuits*

In this section, the first sample circuit, Circuit 1, is used as an example to illustrate the experimental procedures for small combinational circuits. Then, experimental results and the analysis of other sample circuits are given.

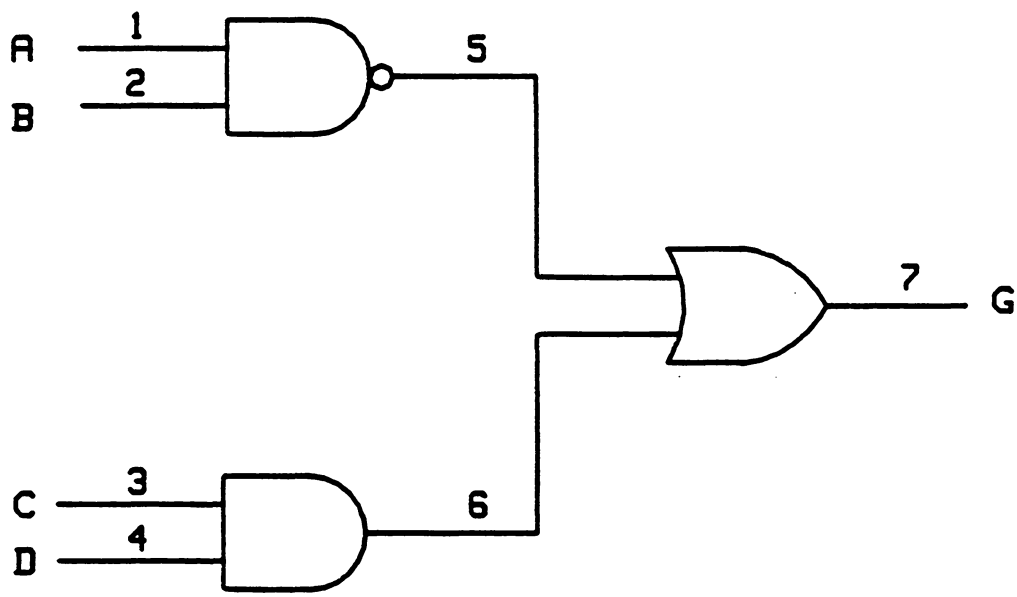
3.2.1 Circuit 1

Consider the circuit shown in Figure 1 on page 16.
The circuit can be modeled by the following substitution statement.

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

This HDL assignment statement contains five micro-operations (two NOT operations, two OR operations, and one AND operation).

To compare the effectiveness of various micro-operation perturbations, we first enumerate the coverage obtained by each input combination. In this case, this was done by using the standard D-algorithm while in some other cases, this was done by using the HILO fault simulator. The coverages for the example circuit of Figure 1 on page 16 is shown in Table 1 on page 19. 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



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

Figure 1. Circuit Diagram for Circuit 1

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 on page 20 shows the set of perturbations applied to the circuit in Figure 1 on page 16. There are five multiple perturbations, three single perturbations, and eight input stuck-at faults.

" Multiple perturbations " means a perturbation rule is applied throughout the good function (G), i.e., all the micro-operations which are subject to the perturbation rule are faulted. For example, fault number 4, OR failed to AND only, perturbs both two OR operations in G and faults them to AND operations. Fault number 1, taking dual function, perturbs all the AND operations in G to OR operations and all the OR operations in G to AND operations. Note that fault number 5 (AND failed to OR only) here actually is a single perturbation since there is only one AND operation in G.

In contrast to multiple perturbations, only one micro-operation is faulted at a time in " single perturbation ". For instance, fault number 6, OR failed to AND, perturbs the first OR operation in G only.

" Input stuck-at fault " forces an input variable to be stuck at one or zero. For example, fault number 9 (A stuck-at-1) forces input variable A to be stuck at 1. Note that the input stuck-at faults, in general sense, are also micro-operation perturbations because they change the micro-operations in G indirectly. For instance, fault number 9 (A stuck-at-1) changes $G = \overline{A} + \overline{B} + CD$ to $G6 = \overline{B} + CD$. The first OR operation and $\overline{A}$ in G are deleted from the faulty function (G6).

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 the set of input combinations that detects the perturbation. Thus, if the ATPG program that derives a test for a micro-operation perturbation selects a random element from the set of all tests for the micro-operation perturbation, the expected value of the coverage obtained is the average coverage listed in the table.

Table 3 on page 21 shows the truth tables for the good function (G) and all of the faulty functions (G1 through G16) from column three of the table in Table 2 on page 20. For example, from Table 3 on page 21 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 42.0%.

From Table 2 on page 20, we observe some micro-operations are better than others. If faults 5, 8, 13, or 15 are used, the expected value for the coverage obtained by selecting a test for the fault is 42.0%. However, if fault 1 (taking dual function) is used, the expected fault coverage is only 18.2 %. The worst fault is fault 4 with only 17.5% average coverage. Also note that AND failed to OR is better than OR failed to AND for both single and multiple perturbations. By examining several circuits, we will look for a generalization that can be made about which micro-operation perturbations are the best.

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 Circuit 1

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A}\,\overline{B}(C + D)$	18.2
2	take complement of G	$G2 = AB(\overline{C} + \overline{D})$	21.0
3	take complement of inputs	$G3 = A + B + \overline{C}\,\overline{D}$	25.7
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	17.5
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	42.0
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B} + CD$	21.0
7	OR -> AND	$G15 = \overline{A} + \overline{B}CD$	22.8
8	AND -> OR	$G16 = \overline{A} + \overline{B} + C + D$	42.0
input stuck-at fault :			
9	A s-a-1	$G6 = \overline{B} + CD$	21.0
10	A s-a-0	$G7 = 1$	39.7
11	B s-a-1	$G8 = \overline{A} + CD$	21.0
12	B s-a-0	$G9 = 1$	39.7
13	C s-a-1	$G10 = \overline{A} + \overline{B} + D$	42.0
14	C s-a-0	$G11 = \overline{A} + \overline{B}$	28.0
15	D s-a-1	$G12 = \overline{A} + \overline{B} + C$	42.0
16	D s-a-0	$G13 = \overline{A} + \overline{B}$	28.0

Table 3. Truth Tables for Micro-Operation Perturbations

COV %	TEST	ABCD	G	G 1	G 2	G 3	G 4	G 5	G 6	G 7	G 8	G 9	G 10	G 11	G 12	G 13	G 14	G 15	G 16
14	0	0000	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
14	1	0001	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
14	2	0010	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
7	3	0011	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
21	4	0100	1	0	1	1	0	1	0	1	1	0	1	1	1	1	1	1	1
21	5	0101	1	0	1	1	0	1	0	1	1	0	1	1	1	1	1	1	1
21	6	0110	1	0	1	1	0	1	0	1	1	0	1	1	1	1	1	1	1
7	7	0111	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
21	8	1000	1	0	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1
21	9	1001	1	0	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1
21	10	1010	1	0	1	1	0	1	0	0	1	1	1	0	1	1	1	1	1
7	11	1011	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
35	12	1100	0	0	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0
42	13	1101	0	0	1	1	0	1	0	0	1	0	1	0	1	1	0	0	0
42	14	1110	0	0	1	1	0	1	0	0	1	0	1	0	1	0	0	1	0
28	15	1111	1	0	0	1	0	1	1	0	1	1	1	1	1	1	0	1	0

3.2.2 Circuit 2

The circuit shown in Figure 2 on page 28 is the NOR/OR implementation of the Circuit 1 circuit. G is still equal to $\overline{A} + \overline{B} + CD$. The experimental results are given in Table 4 on page 29.

Note that the best faults are still faults 5, 8, 13, and 15 and the worst fault is still fault 4. Again, taking dual function is not a good choice. For both single and multiple perturbations, AND failed to OR is better than OR failed to AND. The results match those of Circuit 1 regardless of different implementations.

3.2.3 Circuit 3

The circuit is the AND/NOR implementation of Circuit 1. The circuit is shown in Figure 3 on page 30 and the the results are presented in Table 5 on page 31.

Observe that faults 5, 8, 13, and 15 are still the best faults and AND failed to OR is still better than OR failed to AND. The results coincide with those of Circuit 1 and Circuit 2.

3.2.4 Circuit 4

The OR/NAND implementation of Circuit 1 is shown in Figure 4 on page 32. The results are given in Table 6 on page 33.

Note that in addition to faults 5, 8, 13, and 15, faults 14 and 16 also yield the highest coverages. This means input stuck-at faults do not produce consistent results among different implementations. However, AND failed to OR is still better than OR failed AND and taking dual function is still not a good choice.

3.2.5 Circuit 5

The circuit is the NAND/AND implementation of Circuit 1. The circuit is shown in Figure 5 on page 34. The results are given in Table 7 on page 35. Note that because the coverage of each input combination is the same as that in Circuit 3, the results of this circuit make no differences from those of Circuit 3.

Circuit 1 to Circuit 5 are different implementations of the same function $G = \overline{A} + \overline{B} + CD$. In each implementation, AND failed to OR is better than OR failed to AND and yields the highest average coverage for both single and multiple perturbations. Taking dual function is not a good choice. Although some stuck-at faults of inputs also provide the highest average coverage, there is little consistency among various implementations. For example, C and D stuck-at-1 faults provide the highest average coverage in Circuit 1 while stuck-at-1 and stuck-at-0 faults of C and D all provide the highest average coverage in circuit 4.

3.2.6 Circuit 6

This circuit is the OR/AND implementation of $G = \overline{A}\,\overline{B}(C + D)$. The circuit is shown in Figure 6 on page 36 and it's results are given in Table 8 on page 37.

Observe that faults 4, 6, 14, and 16 provide the highest average coverages, taking dual function is not a good fault, and OR failed to AND is better than AND failed OR for both single and multiple perturbations.

3.2.7 Circuit 7

The AND/NOR implementation of circuit 6 is shown in Figure 7 on page 38. The results are depicted in Table 9 on page 39.

Besides faults 4, 6, 14, and 16, faults 13, 15 are also the best faults. This again indicates input stuck-at faults do not result in consistent conclusions between two different implementations. Like Circuit 6, OR failed to AND is better than AND failed to OR for both single and multiple perturbations.

Circuit 6 and Circuit 7 implement differently the same function $G = \overline{A}\,\overline{B}(C + D)$. OR failed to AND produces the highest average coverage in both cases for both single and multiple perturbations. Again, stuck-at faults of inputs do not provide consistent results between the two cases and taking dual function is not a good choice.

3.2.8 Circuit 8

This circuit is a one-bit full adder, which has three inputs X, Y, and the carry input (Z). The two outputs are the carry output (C) and the sum output (S). The circuit is shown in Figure 8 on page 40. The results for C and S are given in Table 10 on page 41 and Table 11 on page 42.

From C part results, faults 5, 8, 9, 10, 11, 13, and 15 are the best perturbations. The average coverage for taking dual function is zero because taking dual of C function produces the same function as C. Note that AND failed to OR is slightly better than OR failed to AND for both single and multiple perturbations.

From S part results, taking dual function also results in zero average coverage. OR failed to AND is slightly better than AND failed to OR for multiple perturbations. For single perturbations, fault 6, OR failed to AND, is slightly worse than some of the AND failed to OR faults. However, the average of the average coverages of faults 6, 7, and 8 is 27.7%, which is still better than that of

faults 9 through 16 (27.0%). So, we can still say OR failed to AND is better for both single and multiple perturbations.

3.2.9 Circuit 9

The circuit shown in Figure 9 on page 43 implements the carry function of circuit 8. It's results are given in Table 12 on page 44.

From the table, we observe that taking dual function produces zero average coverage and AND failed to OR is better than OR failed to AND for both single and multiple perturbations. Note that fault 15 does not yield the highest average coverage as in C part of Circuit 8. This again indicates there is no consistency about which input stuck-at fault will produce the highest average coverage among different implementations.

3.2.10 Circuit 10

This circuit implements circuit 9 differently and is shown in Figure 10 on page 45. The results are presented in Table 13 on page 46.

Note that AND failed to OR is better than OR failed to AND for both single and multiple perturbations and taking dual function still yield zero average coverage.

Circuit 8 is a one-bit full adder which has two outputs C and S. yields zero coverage in each case. For C part, AND failed to OR provides the highest average coverage for both single and multiple perturbations. For S part, OR failed to AND is slightly better than AND failed to OR. Taking the dual function of C and S (AND -> OR and OR -> AND at the same time) yields zero coverage in each case.

Circuit 9 and Circuit 10 are different implementations of C part function of Circuit 7. AND failed to OR also yields the highest average for both single and multiple perturbations in these two circuits. The results of input stuck-at faults are not consistent among the three circuits.

3.2.11 Circuit 11

This circuit is a two-to-one multiplexer, which is shown in Figure 11 on page 47. The results are given in Table 14 on page 48.

Note that, in this case, taking dual function is among the best perturbations. AND failed to OR provides higher average coverage than OR failed to AND does for both single and multiple perturbations.

3.2.12 Circuit 12

The circuit shown in Figure 12 on page 49 implements $G = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$. The results are presented in Table 15 on page 50.

Faults 4, 6, 12, 13, and 16 are the best perturbations. Note that OR failed to AND is better than AND failed to OR for both single and multiple perturbations. Taking dual function does not yield high average coverage.

3.2.13 Circuit 13

This circuit implements circuit 12 in different way. It's shown in Figure 13 on page 51. It's results are depicted in Table 16 on page 52.

Note that OR failed to AND is again better than AND failed to OR for both single and multiple perturbations. Taking dual function does not produce the highest average coverage. Also note that instead of fault 12 (which yields the highest average coverage in Circuit 12), fault 11 produces the highest fault coverage. This again indicates there is little consistency about input stuck-at faults.

Circuit 12 and Circuit 13 implement the same function $G = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$. In each case, OR failed to AND results in the highest average coverage for both single and multiple perturbations. There is little consistency about input stuck-at faults.

3.2.14 Circuit 14

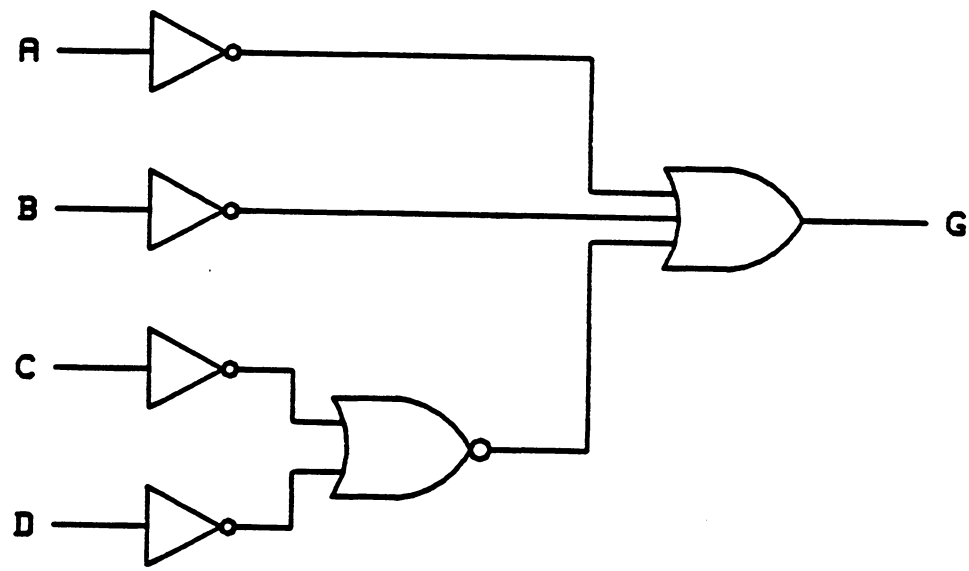
The circuit shown in Figure 14 on page 53 implements $G = AB\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}CD$ The results are given in Table 17 on page 54.

Observe that taking dual function is not a good choice and OR failed to AND is better than AND failed to OR for both single and multiple perturbations.

3.2.15 Circuit 15

The circuit implementing $G = \overline{A} + \overline{B} + CD$ is shown in Figure 15 on page 55. The results are presented in Table 18 on page 56.

In this case, AND failed to OR provides the highest average coverage for both single and multiple perturbations. Taking dual function is almost the worst perturbation.

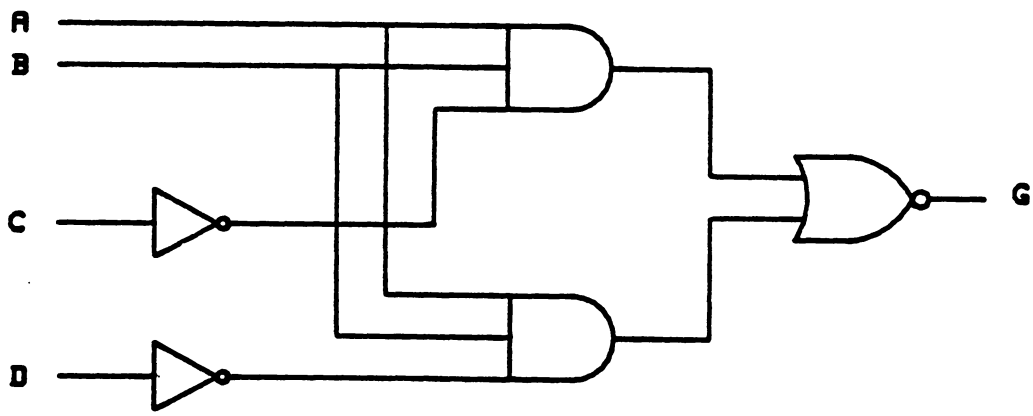


$$G = \overline{A} + \overline{B} + CD$$

Figure 2. Circuit Diagram for Circuit 2

Table 4. Experimental Results for Circuit 2

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A}\,\overline{B}(C + D)$	13.5
2	take complement of G	$G2 = AB(\overline{C} + \overline{D})$	16.3
3	take complement of inputs	$G3 = A + B + \overline{C}\,\overline{D}$	20.8
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	12.1
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	40.0
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B} + CD$	15.0
7	OR -> AND	$G15 = \overline{A} + \overline{B}CD$	18.8
8	AND -> OR	$G16 = \overline{A} + \overline{B} + C + D$	40.0
input stuck-at fault :			
9	A s-a-1	$G6 = \overline{B} + CD$	15.0
10	A s-a-0	$G7 = 1$	36.7
11	B s-a-1	$G8 = \overline{A} + CD$	15.0
12	B s-a-0	$G9 = 1$	36.7
13	C s-a-1	$G10 = \overline{A} + \overline{B} + D$	40.0
14	C s-a-0	$G11 = \overline{A} + \overline{B}$	30.0
15	D s-a-1	$G12 = \overline{A} + \overline{B} + C$	40.0
16	D s-a-0	$G13 = \overline{A} + \overline{B}$	30.0

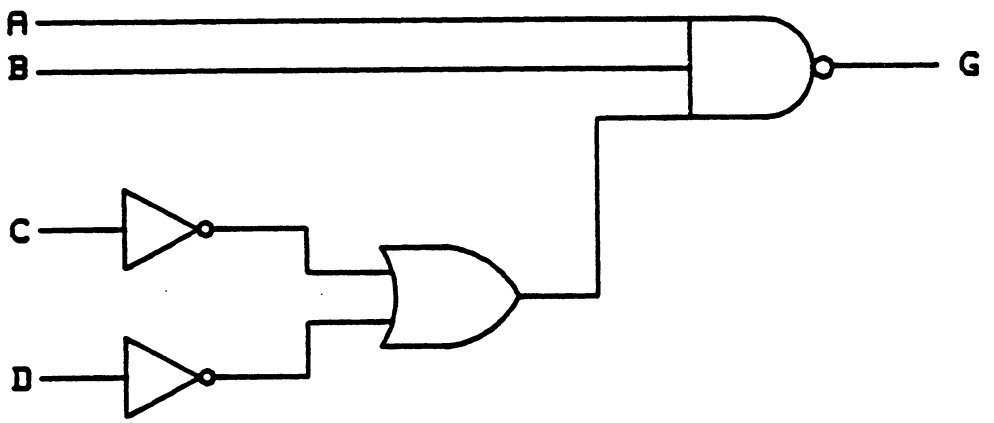


$$G = \overline{A} + \overline{B} + CD$$

Figure 3. Circuit Diagram for Circuit 3

Table 5. Experimental Results for Circuit 3

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A}\,\overline{B}(C + D)$	18.5
2	take complement of G	$G2 = AB(\overline{C} + \overline{D})$	18.4
3	take complement of inputs	$G3 = A + B + \overline{C}\,\overline{D}$	18.3
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	17.4
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	31.0
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B} + CD$	20.3
7	OR -> AND	$G15 = \overline{A} + \overline{B}CD$	22.0
8	AND -> OR	$G16 = \overline{A} + \overline{B} + C + D$	31.0
input stuck-at fault :			
9	A s-a-1	$G6 = \overline{B} + CD$	20.3
10	A s-a-0	$G7 = 1$	24.7
11	B s-a-1	$G8 = \overline{A} + CD$	20.3
12	B s-a-0	$G9 = 1$	24.7
13	C s-a-1	$G10 = \overline{A} + \overline{B} + D$	31.0
14	C s-a-0	$G11 = \overline{A} + \overline{B}$	27.0
15	D s-a-1	$G12 = \overline{A} + \overline{B} + C$	31.0
16	D s-a-0	$G13 = \overline{A} + \overline{B}$	27.0

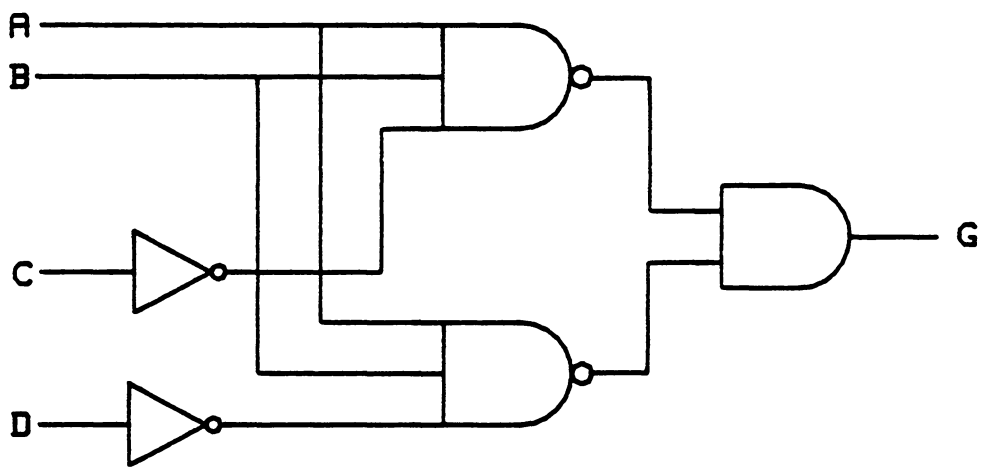


$$G = \overline{A} + \overline{B} + CD$$

Figure 4. Circuit Diagram for Circuit 4

Table 6. Experimental Results for Circuit 4

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A}\,\overline{B}(C + D)$	13.4
2	take complement of G	$G2 = AB(\overline{C} + \overline{D})$	15.8
3	take complement of inputs	$G3 = A + B + \overline{C}\,\overline{D}$	19.8
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	12.2
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	38.0
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B} + CD$	13.0
7	OR -> AND	$G15 = \overline{A} + \overline{B}CD$	19.3
8	AND -> OR	$G16 = \overline{A} + \overline{B} + C + D$	38.0
input stuck-at fault :			
9	A s-a-1	$G6 = \overline{B} + CD$	13.0
10	A s-a-0	$G7 = 1$	33.7
11	B s-a-1	$G8 = \overline{A} + CD$	13.0
12	B s-a-0	$G9 = 1$	33.7
13	C s-a-1	$G10 = \overline{A} + \overline{B} + D$	38.0
14	C s-a-0	$G11 = \overline{A} + \overline{B}$	38.0
15	D s-a-1	$G12 = \overline{A} + \overline{B} + C$	38.0
16	D s-a-0	$G13 = \overline{A} + \overline{B}$	38.0

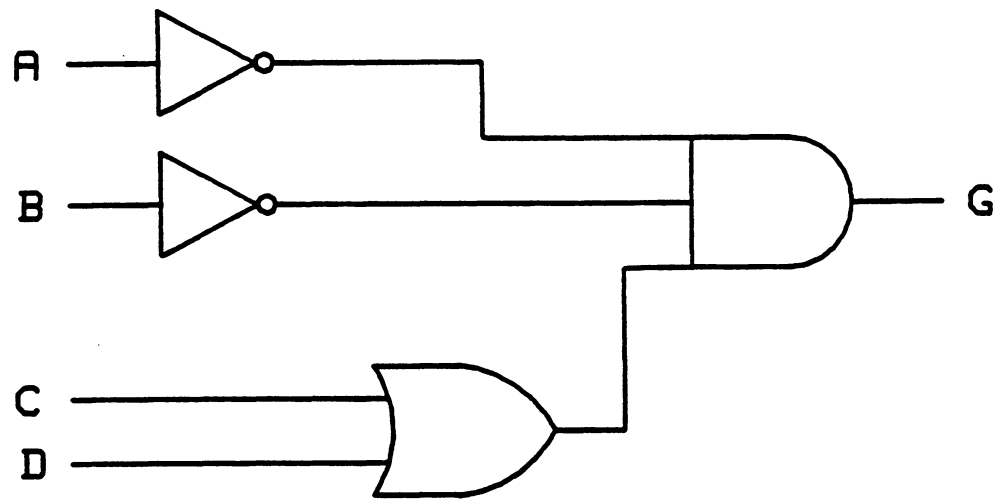


$$G = \overline{A} + \overline{B} + CD$$

Figure 5. Circuit Diagram for Circuit 5

Table 7. Experimental Results for Circuit 5

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A}\,\overline{B}(C + D)$	18.5
2	take complement of G	$G2 = AB(\overline{C} + \overline{D})$	18.4
3	take complement of inputs	$G3 = A + B + \overline{C}\,\overline{D}$	18.3
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	17.4
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	31.0
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B} + CD$	20.3
7	OR -> AND	$G15 = \overline{A} + \overline{B}CD$	22.0
8	AND -> OR	$G16 = \overline{A} + \overline{B} + C + D$	31.0
input stuck-at fault :			
9	A s-a-1	$G6 = \overline{B} + CD$	20.3
10	A s-a-0	$G7 = 1$	24.7
11	B s-a-1	$G8 = \overline{A} + CD$	20.3
12	B s-a-0	$G9 = 1$	24.7
13	C s-a-1	$G10 = \overline{A} + \overline{B} + D$	31.0
14	C s-a-0	$G11 = \overline{A} + \overline{B}$	27.0
15	D s-a-1	$G12 = \overline{A} + \overline{B} + C$	31.0
16	D s-a-0	$G13 = \overline{A} + \overline{B}$	27.0

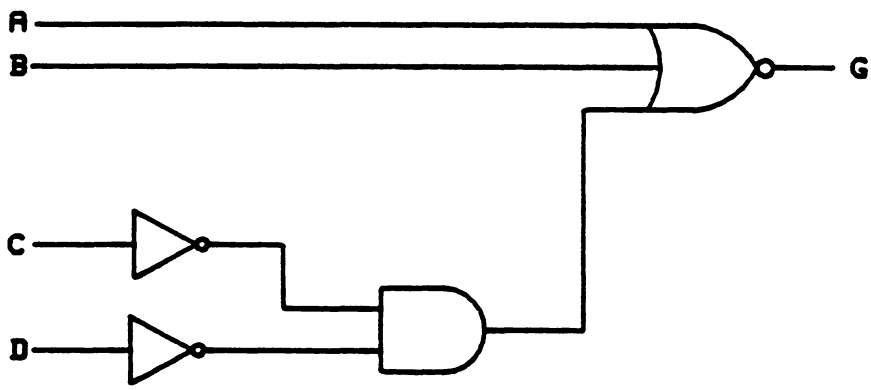


$$G = \overline{A}\,\overline{B}(C + D)$$

Figure 6. Circuit Diagram for Circuit 6

Table 8. Experimental Results for Circuit 6

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A} + \overline{B} + CD$	15.7
2	take complement of G	$G2 = A + B + \overline{C}\,\overline{D}$	18.8
3	take complement of inputs	$G3 = AB(\overline{C} + \overline{D})$	24.0
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	44.0
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	14.1
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B}CD$	44.0
7	AND -> OR	$G15 = \overline{A} + \overline{B}(C + D)$	18.1
8	AND -> OR	$G16 = \overline{A}\,\overline{B} + (C + D)$	15.7
input stuck-at fault :			
9	A s-a-1	$G6 = 0$	42.0
10	A s-a-0	$G7 = \overline{B}(C + D)$	19.0
11	B s-a-1	$G8 = 0$	42.0
12	B s-a-0	$G9 = \overline{A}(C + D)$	19.0
13	C s-a-1	$G10 = \overline{A}\,\overline{B}$	25.0
14	C s-a-0	$G11 = \overline{A}\,\overline{B}D$	44.0
15	D s-a-1	$G12 = \overline{A}\,\overline{B}$	25.0
16	D s-a-0	$G13 = \overline{A}\,\overline{B}C$	44.0

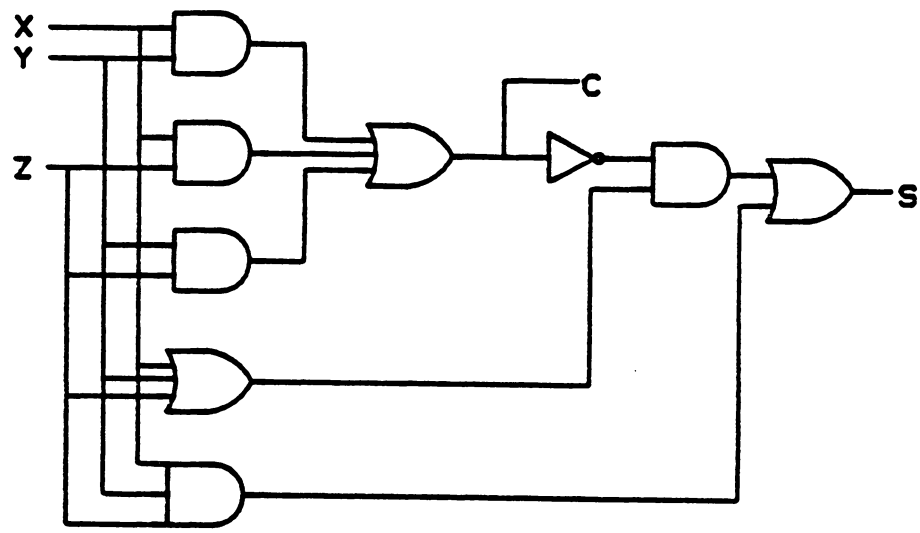


$$G = \overline{A}\,\overline{B}(C + D)$$

Figure 7. Circuit Diagram for Circuit 7

Table 9. Experimental Results for Circuit 7

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = \overline{A} + \overline{B} + CD$	13.4
2	take complement of G	$G2 = A + B + \overline{C}\,\overline{D}$	15.8
3	take complement of inputs	$G3 = AB(\overline{C} + \overline{D})$	19.8
4	OR -> AND only	$G4 = \overline{A}\,\overline{B}CD$	38.0
5	AND -> OR only	$G5 = \overline{A} + \overline{B} + C + D$	12.1
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\,\overline{B}CD$	38.0
7	AND -> OR	$G15 = \overline{A} + \overline{B}(C + D)$	15.3
8	AND -> OR	$G16 = \overline{A}\,\overline{B} + (C + D)$	13.4
input stuck-at fault :			
9	A s-a-1	$G6 = 0$	33.7
10	A s-a-0	$G7 = \overline{B}(C + D)$	13.0
11	B s-a-1	$G8 = 0$	33.7
12	B s-a-0	$G9 = \overline{A}(C + D)$	13.0
13	C s-a-1	$G10 = \overline{A}\,\overline{B}$	38.0
14	C s-a-0	$G11 = \overline{A}\,\overline{B}D$	38.0
15	D s-a-1	$G12 = \overline{A}\,\overline{B}$	38.0
16	D s-a-0	$G13 = \overline{A}\,\overline{B}C$	38.0



$$C = XY + XZ + YZ$$

$$S = XYZ + \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z}$$

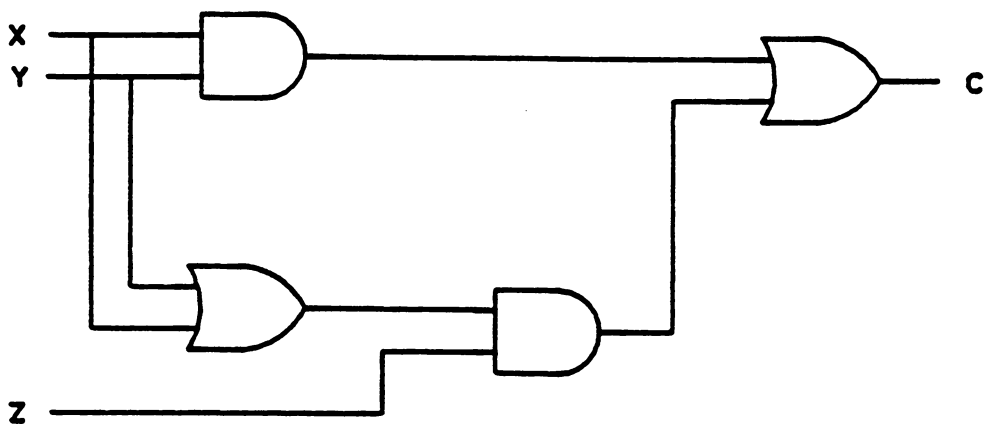
Figure 8. Circuit Diagram for Circuit 8

Table 10. Experimental Results for Circuit 8 - C part

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$C1 = XY + XZ + YZ$	0.0
2	take complement of C	$C2 = \overline{X}\overline{Z} + \overline{X}\overline{Y} + \overline{Y}\overline{Z}$	27.0
3	take complement of inputs	$C3 = \overline{X}\,\overline{Y} + \overline{X}\overline{Z} + \overline{Y}\overline{Z}$	27.0
4	OR -> AND only	$C4 = XYZ$	25.0
5	AND -> OR only	$C5 = X + Y + Z$	29.0
single perturbation :			
6	OR -> AND	$C12 = YZ$	25.0
7	OR -> AND	$C13 = XY$	25.0
8	AND -> OR	$C14 = X + Y$	29.0
9	AND -> OR	$C15 = X + Z$	29.0
10	AND -> OR	$C15 = Y + Z$	29.0
input stuck-at fault :			
11	X s-a-1	$C6 = Y + Z + YZ$	29.0
12	X s-a-0	$C7 = YZ$	25.0
13	Y s-a-1	$C8 = X + XZ + Z$	29.0
14	Y s-a-0	$C9 = XZ$	25.0
15	Z s-a-1	$C10 = XY + X + Y$	29.0
16	Z s-a-0	$C11 = XY$	25.0

Table 11. Experimental Results for Circuit 8 - S part

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$S1 = XYZ + \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z}$	0.0
2	take complement of S	$S2 = \overline{X}\,\overline{Y}\,\overline{Z} + XY\overline{Z} + \overline{X}YZ + X\overline{Y}Z$	27.0
3	take complement of inputs	$S3 = \overline{X}\,\overline{Y}\,\overline{Z} + XY\overline{Z} + X\overline{Y}Z + \overline{X}YZ$	27.0
4	OR -> AND only	$S4 = 0$	27.0
5	AND -> OR only	$S5 = 1$	25.0
single perturbation :			
6	OR -> AND	$S12 = \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z}$	25.0
7	OR -> AND	$S13 = XYZ + X\overline{Y}\,\overline{Z}$	29.0
8	OR -> AND	$S14 = XYZ + \overline{X}\,\overline{Y}Z$	29.0
9	AND -> OR	$S15 = X + YZ + \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z}$	25.0
10	AND -> OR	$S16 = XY + Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z}$	25.0
11	AND -> OR	$S17 = XYZ + \overline{X} + \overline{Y}Z + X\overline{Y}\,\overline{Z}$	27.7
12	AND -> OR	$S18 = \overline{X}\,\overline{Y} + Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z}$	27.7
13	AND -> OR	$S19 = XYZ + \overline{X} + Y\overline{Z} + X\overline{Y}\,\overline{Z}$	27.7
14	AND -> OR	$S20 = XYZ + \overline{X}\,\overline{Y}Z + \overline{X}Y + \overline{Z}$	27.7
15	AND -> OR	$S21 = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X + \overline{Y}\,\overline{Z}$	27.7
16	AND -> OR	$S22 = XYZ + \overline{X}\,\overline{Y}Z + X\overline{Y} + \overline{Z}$	27.7
input stuck-at fault :			
17	X s-a-1	$S6 = YZ + \overline{Y}\,\overline{Z}$	29.0
18	X s-a-0	$S7 = \overline{Y}Z + Y\overline{Z}$	25.0
19	Y s-a-1	$S8 = XZ + \overline{X}\,\overline{Z}$	29.0
20	Y s-a-0	$S9 = \overline{X}Z + X\overline{Z}$	25.0
21	Z s-a-1	$S10 = XY + \overline{X}\,\overline{Y}$	29.0
22	Z s-a-0	$S11 = \overline{X}Y + X\overline{Y}$	25.0

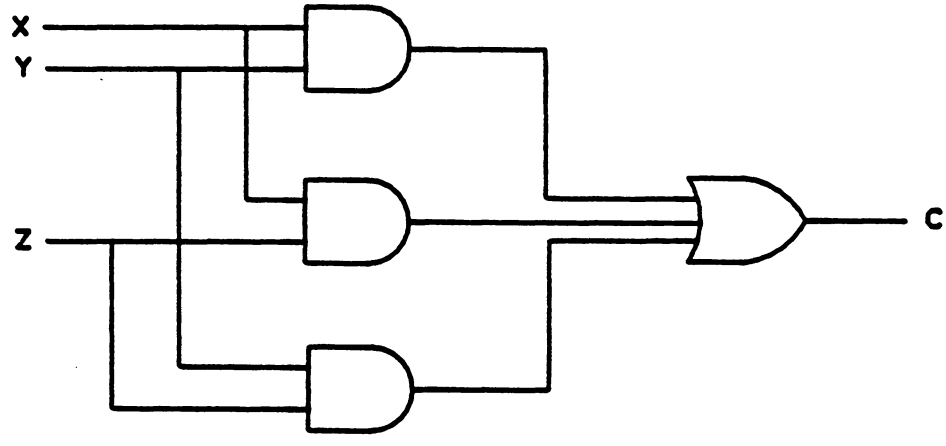


$$C = XY + XZ + YZ$$

Figure 9. Circuit Diagram for Circuit 9

Table 12. Experimental Results for Circuit 9

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$C1 = XY + XZ + YZ$	0.0
2	take complement of C	$C2 = \overline{X}\overline{Z} + \overline{X}\overline{Y} + \overline{Y}\overline{Z}$	23.8
3	take complement of inputs	$C3 = \overline{X}\overline{Y} + \overline{X}\overline{Z} + \overline{Y}\overline{Z}$	23.8
4	OR -> AND only	$C4 = XYZ$	27.0
5	AND -> OR only	$C5 = X + Y + Z$	30.0
single perturbation :			
6	OR -> AND	$C12 = YZ$	27.0
7	OR -> AND	$C13 = XY$	27.0
8	AND -> OR	$C14 = X + Y$	27.0
9	AND -> OR	$C15 = X + Z$	31.5
10	AND -> OR	$C15 = Y + Z$	31.5
input stuck-at fault :			
11	X s-a-1	$C6 = Y + Z + YZ$	31.5
12	X s-a-0	$C7 = YZ$	27.0
13	Y s-a-1	$C8 = X + XZ + Z$	31.5
14	Y s-a-0	$C9 = XZ$	27.0
15	Z s-a-1	$C10 = XY + X + Y$	27.0
16	Z s-a-0	$C11 = XY$	27.0

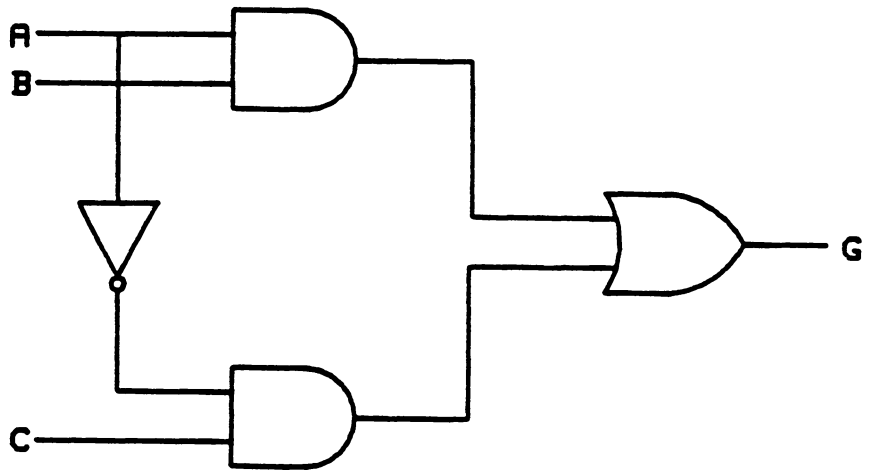


$$C = XY + XZ + YZ$$

Figure 10. Circuit Diagram for Circuit 10

Table 13. Experimental Results for Circuit 10

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$C1 = XY + XZ + YZ$	0.0
2	take complement of C	$C2 = \overline{X}\overline{Z} + \overline{X}\overline{Y} + \overline{Y}\overline{Z}$	28.5
3	take complement of inputs	$C3 = \overline{X}\overline{Y} + \overline{X}\overline{Z} + \overline{Y}\overline{Z}$	28.5
4	OR -> AND only	$C4 = XYZ$	27.0
5	AND -> OR only	$C5 = X + Y + Z$	35.0
single perturbation :			
6	OR -> AND	$C12 = YZ$	27.0
7	OR -> AND	$C13 = XY$	27.0
8	AND -> OR	$C14 = X + Y$	35.0
9	AND -> OR	$C15 = X + Z$	35.0
10	AND -> OR	$C15 = Y + Z$	35.0
input stuck-at fault :			
11	X s-a-1	$C6 = Y + Z + YZ$	35.0
12	X s-a-0	$C7 = YZ$	27.0
13	Y s-a-1	$C8 = X + XZ + Z$	35.0
14	Y s-a-0	$C9 = XZ$	27.0
15	Z s-a-1	$C10 = XY + X + Y$	35.0
16	Z s-a-0	$C11 = XY$	27.0

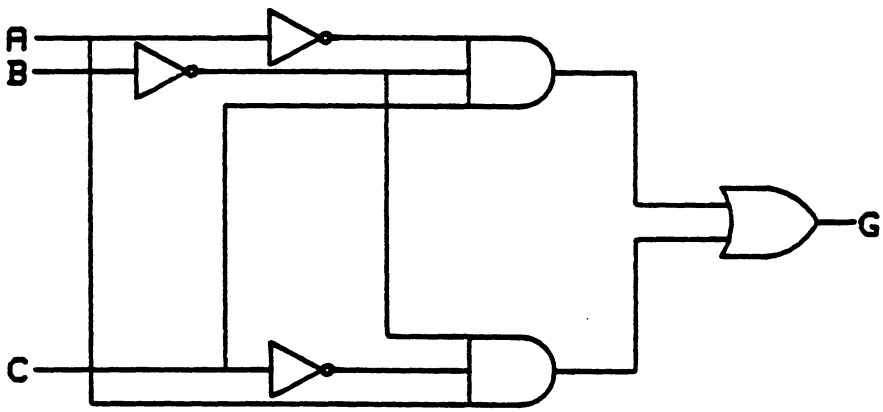


$$G = AB + \overline{A}C$$

Figure 11. Circuit Diagram for Circuit 11

Table 14. Experimental Results for Circuit 11

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = (A + B)(\overline{A} + C)$	33.3
2	take complement of G	$G2 = (\overline{A} + \overline{B})(A + \overline{C})$	27.8
3	take complement of inputs	$G3 = \overline{A}\,\overline{B} + A\overline{C}$	22.2
4	OR -> AND only	$G4 = 0$	25.0
5	AND -> OR only	$G5 = 1$	30.5
single perturbation :			
6	OR -> AND	$G12 = 0$	25.0
7	AND -> OR	$G13 = A + B + C$	33.3
8	AND -> OR	$G14 = \overline{A} + B + C$	33.3
input stuck-at fault :			
9	A s-a-1	$G6 = B$	27.8
10	A s-a-0	$G7 = C$	33.3
11	B s-a-1	$G8 = A + C$	33.3
12	B s-a-0	$G9 = \overline{A}C$	22.2
13	C s-a-1	$G10 = \overline{A} + B$	27.8
14	C s-a-0	$G11 = AB$	27.8

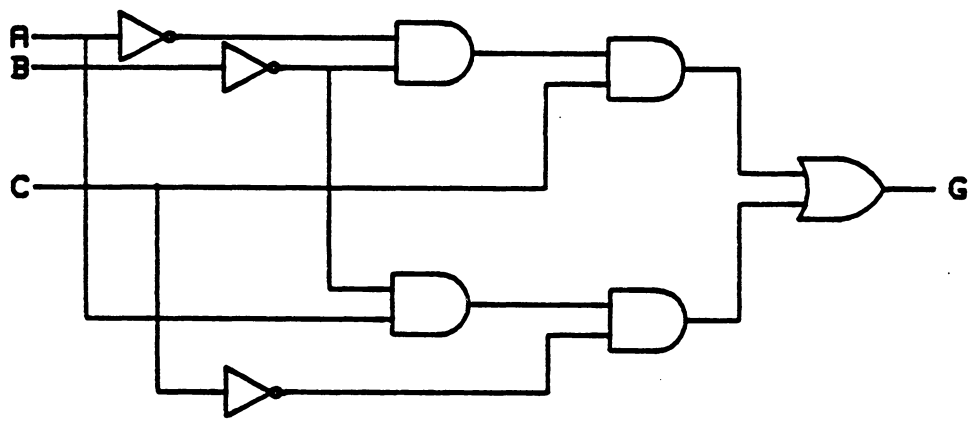


$$G = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$$

Figure 12. Circuit Diagram for Circuit 12

Table 15. Experimental Results for Circuit 12

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = (\overline{A} + \overline{B} + C)(A + \overline{B} + \overline{C})$	20.8
2	take complement of G	$G2 = (A + B + \overline{C})(\overline{A} + B + C)$	25.0
3	take complement of inputs	$G3 = AB\overline{C} + \overline{A}BC$	29.1
4	OR -> AND only	$G4 = 0$	41.6
5	AND -> OR only	$G5 = 1$	19.4
single perturbation :			
6	OR -> AND	$G12 = 0$	41.6
7	AND -> OR	$G13 = \overline{A} + \overline{B}C + A\overline{B}\overline{C}$	22.9
8	AND -> OR	$G14 = \overline{A}\overline{B} + C + A\overline{B}\overline{C}$	22.9
9	AND -> OR	$G15 = \overline{A}\overline{B}C + A + \overline{B}\overline{C}$	22.9
10	AND -> OR	$G16 = \overline{A}\overline{B}C + A\overline{B} + \overline{C}$	22.9
input stuck-at fault :			
11	A s-a-1	$G6 = \overline{B}\overline{C}$	33.3
12	A s-a-0	$G7 = \overline{B}C$	41.6
13	B s-a-1	$G8 = 0$	41.6
14	B s-a-0	$G9 = \overline{A}C + A\overline{C}$	16.6
15	C s-a-1	$G10 = \overline{A}\overline{B}$	33.3
16	C s-a-0	$G11 = A\overline{B}$	41.6

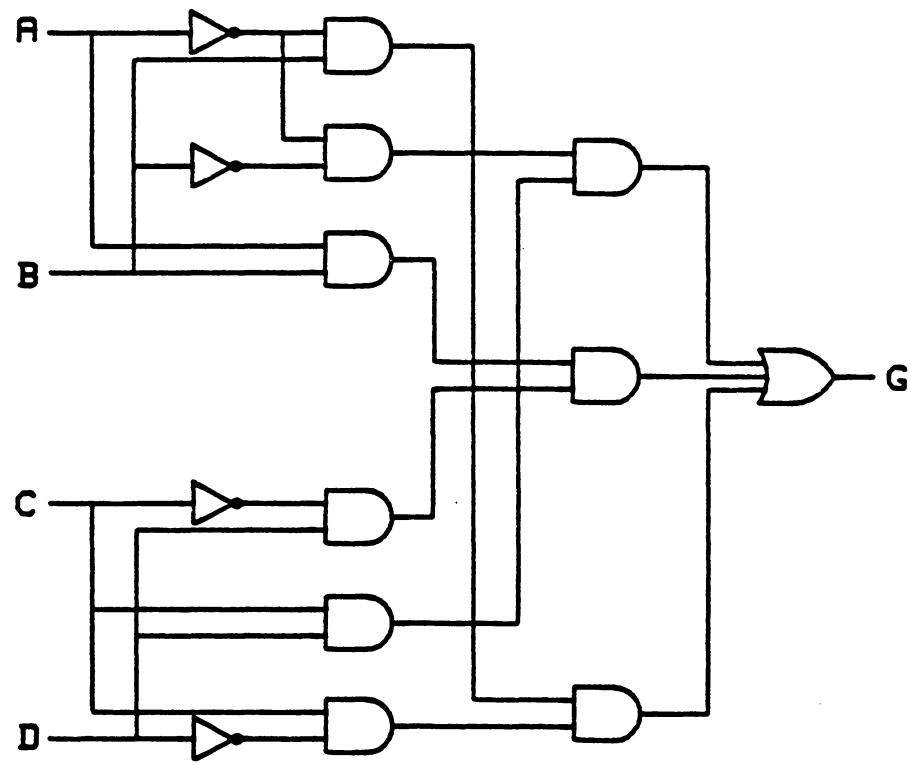


$$G = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$$

Figure 13. Circuit Diagram for Circuit 13

Table 16. Experimental Results for Circuit 13

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = (\overline{A} + \overline{B} + C)(A + \overline{B} + \overline{C})$	28.5
2	take complement of G	$G2 = (A + B + \overline{C})(\overline{A} + B + C)$	30.3
3	take complement of inputs	$G3 = AB\overline{C} + \overline{A}BC$	32.1
4	OR -> AND only	$G4 = 0$	32.1
5	AND -> OR only	$G5 = 1$	29.7
single perturbation :			
6	OR -> AND	$G12 = 0$	32.1
7	AND -> OR	$G13 = \overline{A} + \overline{B}C + A\overline{B}\overline{C}$	30.3
8	AND -> OR	$G14 = \overline{A}\overline{B} + C + A\overline{B}\overline{C}$	30.3
9	AND -> OR	$G15 = \overline{A}\overline{B}C + A + \overline{B}\overline{C}$	28.5
10	AND -> OR	$G16 = \overline{A}\overline{B}C + A\overline{B} + \overline{C}$	28.5
input stuck-at fault :			
11	A s-a-1	$G6 = \overline{B}\overline{C}$	32.1
12	A s-a-0	$G7 = \overline{B}C$	28.5
13	B s-a-1	$G8 = 0$	32.1
14	B s-a-0	$G9 = \overline{A}C + A\overline{C}$	32.1
15	C s-a-1	$G10 = \overline{A}\overline{B}$	28.5
16	C s-a-0	$G11 = A\overline{B}$	32.1

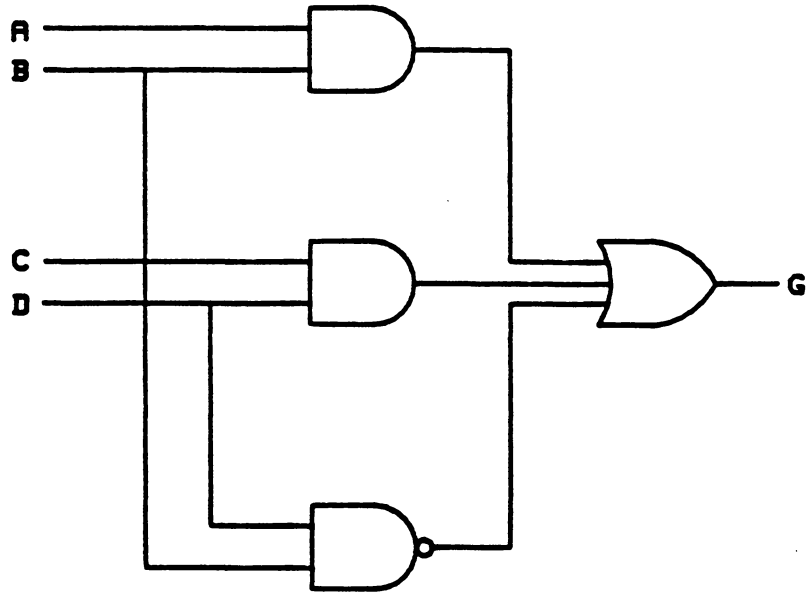


$$G = AB\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}CD$$

Figure 14. Circuit Diagram for Circuit 14

Table 17. Experimental Results for Circuit 14

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = 1$	17.6
2	take complement of G	$G2 = \overline{B}\overline{D} + \overline{C}\overline{D} + \overline{A}\overline{C} + A\overline{B} + AC + \overline{A}BD$	18.7
3	take complement of inputs	$G3 = \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}D + AB\overline{C}\overline{D}$	22.9
4	OR -> AND only	$G4 = 0$	23.6
5	AND -> OR only	$G5 = 1$	17.6
single perturbation :			
6	OR -> AND	$G14 = \overline{A}\overline{B}CD$	22.9
7	OR -> AND	$G15 = AB\overline{C}D$	25.0
8	AND -> OR	$G16 = A + B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\overline{B}CD$	19.3
9	AND -> OR	$G17 = AB + \overline{C}D + \overline{A}BC\overline{D} + \overline{A}\overline{B}CD$	19.4
10	AND -> OR	$G18 = AB\overline{C} + D + \overline{A}BC\overline{D} + \overline{A}\overline{B}CD$	18.4
11	AND -> OR	$G19 = AB\overline{C}D + \overline{A} + BC\overline{D} + \overline{A}\overline{B}CD$	17.8
12	AND -> OR	$G20 = AB\overline{C}D + \overline{A}B + C\overline{D} + \overline{A}\overline{B}CD$	20.1
13	AND -> OR	$G21 = AB\overline{C}D + \overline{A}BC + \overline{D} + \overline{A}\overline{B}CD$	17.1
14	AND -> OR	$G22 = AB\overline{C}D + \overline{A}BC\overline{D} + \overline{A} + \overline{B}CD$	17.8
15	AND -> OR	$G23 = AB\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\overline{B} + CD$	14.6
16	AND -> OR	$G24 = AB\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\overline{B}C + D$	17.8
input stuck-at fault :			
17	A s-a-1	$G6 = B\overline{C}D$	25.0
18	A s-a-0	$G7 = BC\overline{D} + \overline{B}CD$	23.6
19	B s-a-1	$G8 = A\overline{C}D + \overline{A}C\overline{D}$	22.2
20	B s-a-0	$G9 = \overline{A}CD$	19.4
21	C s-a-1	$G10 = \overline{A}B\overline{D} + \overline{A}\overline{B}D$	18.0
22	C s-a-0	$G11 = ABD$	18.0
23	D s-a-1	$G12 = AB\overline{C} + \overline{A}\overline{B}C$	23.6
24	D s-a-0	$G13 = \overline{A}BC$	19.4



$$G = AB + CD + \overline{B} + \overline{D}$$

Figure 15. Circuit Diagram for Circuit 15

Table 18. Experimental Results for Circuit 15

FAULT NO.	PERTURBATION	FAULTY FUNCTION	AVE. % COVERAGE
multiple perturbations :			
1	take dual function	$G1 = (A + B)(C + D)\overline{B}\,\overline{D}$	9.3
2	take complement of G	$G2 = (\overline{A} + \overline{B})(\overline{C} + \overline{D})BD$	10.4
3	take complement of inputs	$G3 = \overline{A}\,\overline{B} + \overline{C}\,\overline{D} + B + D$	18.5
4	OR -> AND only	$G4 = 0$	9.2
5	AND -> OR only	$G5 = 1$	29.0
single perturbation :			
6	OR -> AND	$G14 = ABCD + \overline{B} + \overline{D}$	17.0
7	OR -> AND	$G15 = AB + CD\overline{B} + \overline{D}$	14.3
8	OR -> AND	$G16 = AB + CD + \overline{B}\,\overline{D}$	10.5
9	AND -> OR	$G17 = 1$	29.0
10	AND -> OR	$G18 = 1$	29.0
input stuck-at fault :			
11	A s-a-1	$G6 = 1$	29.0
12	A s-a-0	$G7 = CD + \overline{B} + \overline{D}$	17
13	B s-a-1	$G8 = A + CD + \overline{D}$	13.0
14	B s-a-0	$G9 = 1$	29.0
15	C s-a-1	$G10 = 1$	29.0
16	C s-a-0	$G11 = AB + \overline{B} + \overline{D}$	17.0
17	D s-a-1	$G12 = AB + C + \overline{B}$	13.0
18	D s-a-0	$G13 = 1$	29.0

3.3 *Analysis Summary for Small Combinational Circuits*

From the experimental results and analysis of previous section, 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.

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 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.

4. The results of input stuck-at faults show no regularity. That is, it is hard to predict which input stuck-at fault will provide good average coverage.

4.0 Evaluation Method, Experiments and Analysis for Large Circuits

This chapter describes the evaluation method for various micro-operation perturbations in large combinational circuits. For large combinational circuits, a statistical method was employed to estimate the average coverage for a micro-operation perturbation in the form of 95% confidence intervals. The confidence interval for each micro-operation perturbation is then used as the comparison metric. After the evaluation method is described, a second example using SN7483a adder is given to illustrate the experimental procedures for large combinational circuits. The results and the analysis of SN74181 ALU are also presented. Finally, the analysis summary of the results and some suggestions for improvement are made.

4.1 *Evaluation Method for Large Combinational Circuits*

From the previous chapter 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 [29] 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,

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

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

$\mathbf{t_{0.05}(df)}$ 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 (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 [2,30], we adopted GSP2 [20] 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 fault 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.

Note that usually we need to sample more than one times to achieve the 95% confidence interval. By comparing 95% confidence intervals of various micro-operation fault models, we can decide which one is the best choice.

4.2 *Experiments for Large Combinational Circuits*

This section describes the experimental procedures for large combinational circuits. Sample circuits are modeled in GSP2[20] on a DEC VAX running VMS. The random sampling technique is coded in Pascal and HP-UX Bourne Shell Script on a HP 550 computer running HP-UX, which is the Hewlett-Packard version of UNIX. The procedure of calculating the 95% confidence intervals is automated by a Bourne Shell program.

An example using SN7483a adder is given to demonstrate the experimental procedures and then the experimental results of SN74181 ALU are presented. GSP2 programs for the experiments of SN7483a adder are given in Appendix A, GSP2 programs for the experiments of SN74181 ALU are presented in Appendix B, and the Pascal and HP-UX Script programs for calculating the coverage intervals are given in Appendix C.

In order not only to be able to tell which micro-operation perturbations are the best but also to see the accumulated coverage of multiple tests, we divided an experiment into the following four parts.

1. A part : In this part of experiment, we try to decide which micro-operation perturbations are the best in terms of higher 95% confidence interval. This goal is the same as that in experiments for small combinational circuits stated in last chapter. The only difference is that the average coverage of a test set for a micro-operation perturbation is used as the comparison metric for small combinational circuits while the 95% confidence coverage interval for a micro-operation perturbation is employed as a measure of effectiveness for large combinational circuits.

   For a micro-operation perturbation, we first find its test set by applying exhaustive input combinations to both good and faulty modules. Second, we randomly sample one test from the test set and input the test into the HILO fault simulator to get the gate level stuck-at fault coverage for the test. At least two samplings are needed to calculate the 95% confidence cov-

erage interval for the micro-operation perturbation. If an acceptable 95% confidence interval has been achieved, i.e., the resolution of the interval is good ($d < 0.02$), the procedure is terminated; otherwise, more samplings are added until the confidence interval is acceptable. Note that usually we need to sample more than one times to achieve the 95% confidence interval. Various micro-operation perturbations are experimented to determine which perturbations are the best in terms of higher 95% confidence interval.

The next three parts deal with the accumulation effect of multiple tests. Note that the 95% confidence interval in A part experiment represents the expected coverage range for only one test picked randomly from the test set for a micro-operation perturbation. The next three parts try to see the effect of multiple tests.

2. B part : From the A part experiment, we will get some idea about which micro-operation is the best. In this part, The best micro-operation perturbation for each micro-operation is chosen to fault each micro-operation. One test is then picked from the test set for each best micro-operation perturbation and all the tests picked are input to the HILO fault simulator to get the accumulated gate level stuck-at fault coverage for all the micro-operations. The sampling process may need to be repeated several times to achieve the 95% confidence coverage interval. For example, if there are ten micro-operations in a chip level description of a circuit, we inject the best perturbation for each micro-operation, one at a time, to obtain their respective test set. Then, one test is picked from each of the ten test sets and all the ten tests are input to the HILO fault simulator to get the accumulated gate level stuck-at fault coverage for these ten micro-operations. Finally, the sampling process (each time pick ten tests, one from each of the ten test sets) may need to be repeated several times to achieve the 95% confidence interval for the ten micro-operations.

3. C part : Like B part, the best micro-operation perturbation for each micro-operations is still used to fault each micro-operation. But, in this part, multiple tests are picked from the test set for each best micro-operation perturbation. For example, if there are ten micro-operations in a chip level HDL description, we inject the best perturbation for each micro-operation, one

at a time, to obtain their respective test set. Then, multiple tests (say, four tests) are picked from each of the ten test sets and all the forty tests are input to the HILO fault simulator to get the accumulated coverage for the ten micro-operations. Again, the sampling process (each time pick forty tests, four from each of the ten micro-operations) may need to be repeated several times to achieve the 95% confidence interval for the ten micro-operations.

4. D part : Instead of using the best micro-operation perturbation for each micro-operation in B and C parts, we use multiple micro-operation perturbations for each micro-operation. That is, for each micro-operation in a chip level description of a circuit, all possible perturbations are considered and one test set is obtained from each perturbation. And then, one test is picked from each test set and all the tests are used to perform the fault simulation to get the accumulated coverage. For example, if there are ten micro-operations in a chip level description of a circuit and there are four possible ways of perturbation for each of the ten micro-operations, a test set is obtained for each possible perturbations of each micro-operation (four test sets for each micro-operation). Then, a test is picked from each test set and a total of forty tests are input to HILO fault simulator to get the accumulated coverage. Finally, the sampling process (each time pick forty tests, four from each of the ten micro-operations) may need to be repeated to achieve the 95% confidence interval for the ten micro-operations.

4.2.1 Experiments with SN7483a Carry-Look-Ahead adder

1. A part :

The chip level GSP2 description for the 4-bit carry-look-ahead adder (SN7483a) is given Appendix A.4. It contains two ADD micro-operations. The first one is as follows :

```
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 be 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);
```

Figure 16 on page 66 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 Figure 17 on page 68. 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

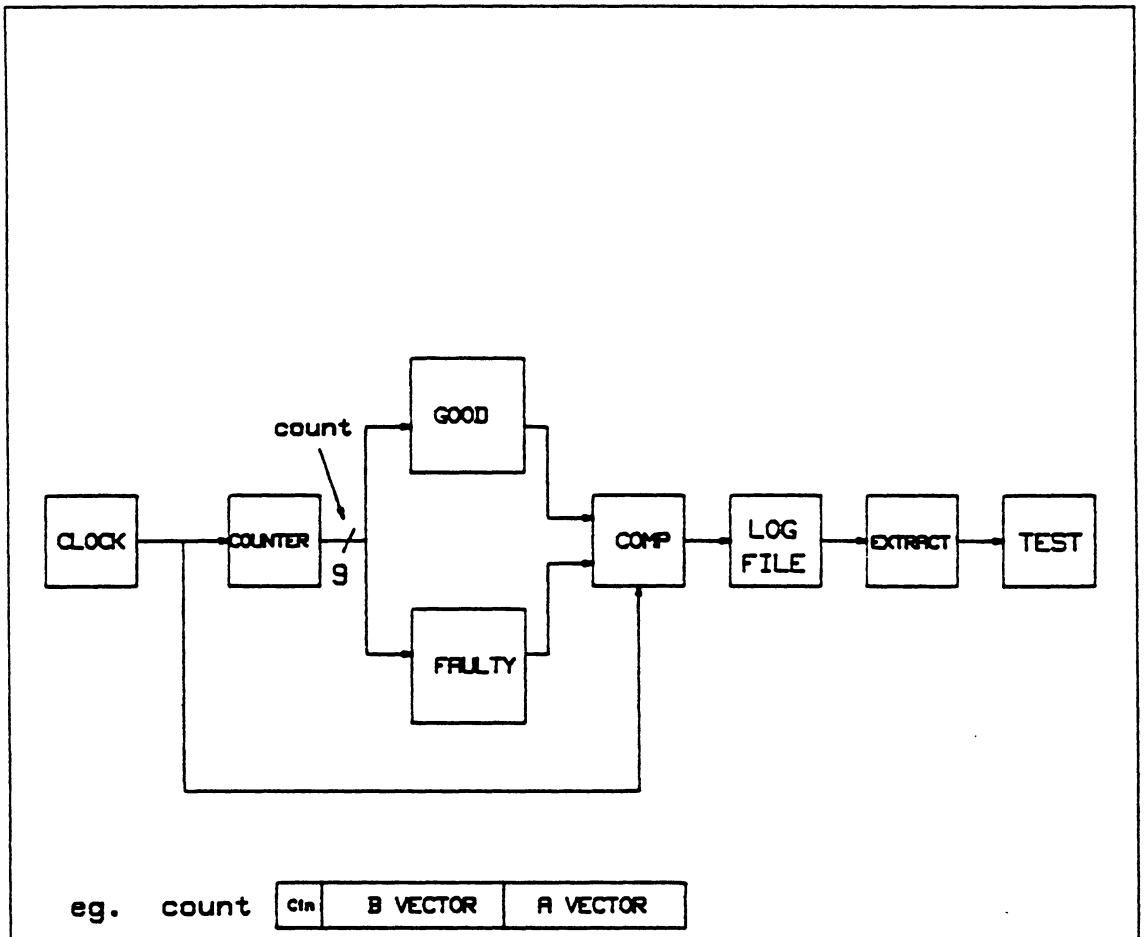


Figure 16. Using GSP2 to Obtain the Test Set

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 an HP-UX Bourne Shell program, is the master of the system. It automates the random sampling technique.

Note that since in our case the test set is usually larger than the not-test set, the EXTRACT is used to extract the not-test set from the LOG FILE. The not-test set is then input to the WRITE.P module, which converts the not-test set to the test set.

The experimental results are summarized in Table 19 on page 71. Faults 1-1, 1-2, 1-3, and 1-4 are micro-operation substitution faults for the first ADD micro-operation in the chip level GSP2 description of the SN7483a adder. Faults 1-5, 1-6, 1-7, and 1-8 are micro-operation stuck-at faults in which the output of the ADD operation, i.e., int_result1, is stuck at some constant patterns. Faults 1-9 through 1-18 are also micro-operation stuck-at faults in which single bits of int_result1 are stuck at either 1 or 0. The numbers of tests which can detect some micro-operation fault are listed in the third column. The fourth column shows the sampling iterations needed to achieve the 95% confidence interval for some perturbation. The 95% confidence coverage interval for each fault is depicted in the last column.

Observe that all of the micro-operation faults results in about the same 95% confidence coverage interval. This means, in this case, there is no best perturbation among those micro-operation perturbations. The experimental results of the second micro-operation are also given in Table 20 on page 72. The table also shows similar 95% confidence coverage intervals and no best perturbation.

2. B part

In this part, the best micro-operation perturbation for each micro-operation is chosen to fault each micro-operation. Since from A part, it seems there is no best perturbations for the two ADD micro-operations of the adder, we perform B part experiment, respectively, for each possible perturbation of the ADD operations. For example, the two ADD operations are

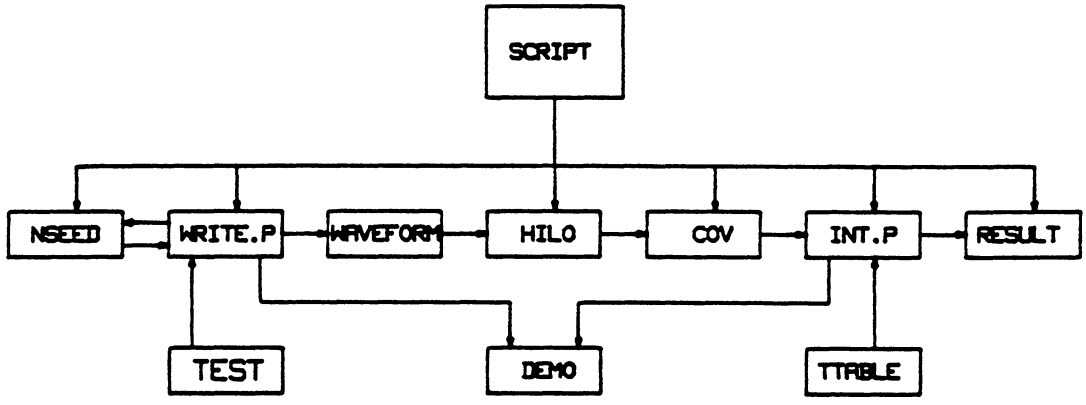


Figure 17. HP-UX Script to Automate the Experimental Procedure

faulted to SUB operations, one at a time, to get their respective test set. Then, one test is picked from each test set and a total of two tests are input to the HILO fault simulator. The sampling procedure is repeated 36 times to achieve the 95% confidence interval.

Similar experiments are performed for ADD failed to AND, OR and EXOR. The results are shown in Table 21 on page 73. The table indicates similar coverage intervals and therefore confirms there is no best micro-operation perturbation among the four possible perturbations for the ADD operation. Also note that the coverage intervals are better than that of A part. This is because the coverage intervals here are for two tests while the coverage intervals of A part are for only one test.

3. C part:

The experiment of this part is similar to that of B part except that we pick four tests instead of one test from each test set for the best micro-operation perturbations. For example, after the test sets for the two ADD failed to SUB perturbations are obtained, four tests are picked from each test set and a total of eight tests are input to HILO fault simulator.

Again, since from A part, it seems there is no best perturbation for the ADD operation, similar experiments are performed for ADD failed to AND, OR and EXOR. The results are depicted in Table 22 on page 74. Observe that similar coverage intervals are produced and therefore there is no best perturbation. The coverage intervals are better than those of B part because the coverage intervals here are for eight tests while the coverage intervals of B part are for only two tests.

4. D part :

In this part, multiple micro-operation perturbations are used for each micro-operation. That is, each ADD operation in the GSP2 description of SN7483a adder is faulted to SUB, AND, OR, and EXOR, one at a time. After the test set for each perturbation is obtained, one test is picked from each test set and a total of eight tests are input to HILO fault simulator. It takes 19 sampling iterations to achieve the 95% confidence interval. The results are shown in

Table 23 on page 75. The resulting coverage interval is similar to that of C part since they are all for eight tests.

Table 19. A Part Results for SN7483a First ADD Micro-operation

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

Table 20. A Part Results for SN7483a Second ADD Micro-operation

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
micro-operation substitution fault :				
2-1	ADD -> SUB	256	4	[0.359, 0.398]
2-2	ADD -> AND	511	10	[0.362, 0.399]
2-3	ADD -> OR	128	9	[0.364, 0.402]
2-4	ADD -> EXOR	128	3	[0.357, 0.386]
int_result1 stuck-at faults :				
2-5	ADD -> 01111	481	8	[0.355, 0.391]
2-6	ADD -> 10000	481	9	[0.368, 0.407]
2-7	ADD -> 11100	505	16	[0.363, 0.402]
2-8	ADD -> 00011	505	10	[0.360, 0.397]
2-9	bit 0 s-a-0	256	12	[0.383, 0.423]
2-10	bit 0 s-a-1	255	9	[0.352, 0.390]
2-11	bit 1 s-a-0	256	4	[0.369, 0.403]
2-12	bit 1 s-a-1	255	5	[0.347, 0.376]
2-13	bit 2 s-a-0	256	4	[0.359, 0.398]
2-14	bit 2 s-a-1	255	3	[0.345, 0.376]
2-15	bit 3 s-a-0	256	9	[0.369, 0.409]
2-16	bit 3 s-a-1	255	2	[0.368, 0.368]
2-17	bit 4 s-a-0	256	6	[0.361, 0.400]
2-18	bit 4 s-a-1	255	6	[0.384, 0.416]

Table 21. B Part Results for SN7483A

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
1-1 2-1	ADD -> SUB ADD -> SUB	36	[0.565, 0.604]
1-2 2-2	ADD -> AND ADD -> AND	31	[0.565, 0.605]
1-3 2-3	ADD -> OR ADD -> OR	24	[0.550, 0.589]
1-4 2.4	ADD -> EXOR ADD -> EXOR	36	[0.561, 0.600]

Table 22. C Part Results for SN7483A

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
1-1 2-1	ADD -> SUB ADD -> SUB	9	[0.855, 0.892]
1-2 2-2	ADD -> AND ADD -> AND	5	[0.875, 0.909]
1-3 2-3	ADD -> OR ADD -> OR	19	[0.848, 0.887]
1-4 2-4	ADD -> EXOR ADD -> EXOR	20	[0.854, 0.894]

Table 23. D Part Results for SN7483A

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
1-1	ADD -> SUB	19	[0.854, 0.893]
1-2	ADD -> AND		
1-3	ADD -> OR		
1-4	ADD -> EXOR		
2-1	ADD -> SUB		
2-2	ADD -> AND		
2-3	ADD -> OR		
2-4	ADD -> EXOR		

4.2.2 Experiments with SN74181 ALU

SN74181 ALU is a good circuit to experiment because it exercises various logic function and arithmetic functions. It's GSP2 description is given in Appendix B.4.

1. A part:

   Thirty-five micro-operations in various logic and arithmetic statements are experimented with micro-operation substitution faults. Four among them (micro-operations 11, 14, 15, 19) are also experimented with micro-operation stuck-at faults. The results are summarized in Table 24 on page 78 to Table 30 on page 84.

   Observe that for each micro-operation, all perturbations result in similar 95% confidence coverage intervals. This indicates that there is no best micro-operation perturbations for AND, OR, ADD, SUB, and EXOR.

2. B part :

   Since all perturbations produced similar coverage intervals from A part, micro-operation 11, 14, 15, and 19 are chosen to fault to their duals. Four tests, one for each fault, are randomly sampled to enumerate the 95% confidence interval. It takes 5 sampling iterations to achieve the 95% confidence interval. The results are given in Table 31 on page 85.

   Note that the coverage interval is better than that of A part because the coverage interval is for 4 tests while the coverage intervals of A part are for only one test.

3. C part :

   Micro-operations 11, 14, 15, and 19 are faulted to their duals. Four tests are picked for each perturbation. Total 16 tests are sampled in Table 32 on page 86.

   Note that because the coverage interval is for 16 tests, it's better than the coverage interval of B part, which is for only 4 tests.

4. D part :

Four micro-operation perturbations are used for each of the micro-operations 11, 14, 15, and 19. One test is picked from each micro-operation perturbation. Total 16 tests are sampled in Table 33 on page 87.

Observe that the interval is similar to that of C part because both intervals are for 16 tests.

Table 24. A Part Results for SN74181 ALU

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
1-1	OR -> AND	480	5	[0.270, 0.298]
1-2	OR -> EXOR	350	7	[0.253, 0.291]
1-3	OR -> ADD	350	12	[0.235, 0.271]
1-4	OR -> SUB	450	3	[0.250, 0.282]
2-1	AND -> OR	480	11	[0.264, 0.301]
2-2	AND -> EXOR	510	12	[0.265, 0.304]
2-3	AND -> ADD	510	5	[0.254, 0.294]
2-4	AND -> SUB	510	6	[0.264, 0.304]
3-1	AND -> OR	480	6	[0.276, 0.316]
3-2	AND -> EXOR	510	4	[0.294, 0.332]
3-3	AND -> ADD	510	4	[0.229, 0.264]
3-4	AND -> SUB	510	4	[0.292, 0.324]
4-1	EXOR -> OR	350	3	[0.321, 0.341]
4-2	EXOR -> AND	510	4	[0.320, 0.346]
4-3	EXOR -> ADD	296	4	[0.311, 0.341]
4-4	EXOR -> SUB	296	3	[0.318, 0.338]
5-1	AND -> OR	480	4	[0.258, 0.295]
5-2	AND -> EXOR	510	4	[0.272, 0.304]
5-3	AND -> ADD	510	3	[0.264, 0.296]
5-4	AND -> SUB	510	5	[0.263, 0.296]
6-1	OR -> AND	480	5	[0.269, 0.299]
6-2	OR -> EXOR	350	4	[0.265, 0.302]
6-3	OR -> ADD	350	2	[0.300, 0.300]
6-4	OR -> SUB	450	4	[0.272, 0.304]
7-1	EXOR -> OR	350	3	[0.288, 0.308]
7-2	EXOR -> AND	510	4	[0.284, 0.312]
7-3	EXOR -> ADD	296	3	[0.279, 0.316]
7-4	EXOR -> SUB	296	3	[0.298, 0.318]

Table 25. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
8-1	AND -> OR	480	3	[0.248, 0.268]
8-2	AND -> EXOR	510	2	[0.260, 0.260]
8-3	AND -> ADD	510	4	[0.251, 0.288]
8-4	AND -> SUB	510	4	[0.261, 0.282]
9-1	OR -> AND	480	6	[0.270, 0.305]
9-2	OR -> EXOR	482	3	[0.264, 0.296]
9-3	OR -> ADD	482	4	[0.271, 0.298]
9-4	OR -> SUB	450	7	[0.274, 0.306]
10-1	OR -> AND	480	3	[0.290, 0.322]
10-2	OR -> EXOR	350	9	[0.271, 0.307]
10-3	OR -> ADD	450	5	[0.270, 0.308]
10-4	OR -> SUB	450	4	[0.282, 0.317]
11-1	OR -> AND	240	4	[0.301, 0.328]
11-2	OR -> EXOR	175	3	[0.310, 0.347]
11-3	OR -> ADD	175	8	[0.285, 0.324]
11-4	OR -> SUB	225	3	[0.299, 0.336]
output of OR operation stuck-at faults:				
11-5	bit 0 s-a-0	192	4	[0.298, 0.338]
11-6	bit 0 s-a-1	64	2	[0.300, 0.300]
11-7	bit 1 s-a-0	192	3	[0.328, 0.348]
11-8	bit 1 s-a-1	64	6	[0.275, 0.311]
11-9	bit 2 s-a-0	192	3	[0.325, 0.345]
11-10	bit 2 s-a-1	64	8	[0.278, 0.313]
11-11	bit 3 s-a-0	192	7	[0.287, 0.323]
11-12	bit 3 s-a-1	64	7	[0.303, 0.342]
12-1	OR -> AND	240	6	[0.309, 0.346]
12-2	OR -> EXOR	176	7	[0.303, 0.336]
12-3	OR -> ADD	176	3	[0.318, 0.338]
12-4	OR -> SUB	225	8	[0.297, 0.332]

Table 26. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
13-1	ADD -> SUB	148	4	[0.310, 0.346]
13-2	ADD -> AND	240	21	[0.297, 0.337]
13-3	ADD -> OR	240	6	[0.291, 0.324]
13-4	ADD -> EXOR	148	4	[0.313, 0.349]
14-1	AND -> OR	240	54	[0.336, 0.368]
14-2	AND -> EXOR	255	22	[0.288, 0.327]
14-3	AND -> ADD	255	15	[0.293, 0.332]
14-4	AND -> SUB	255	15	[0.289, 0.328]
output of AND operation stuck-at faults:				
14-5	bit 0 s-a-0	64	10	[0.315, 0.353]
14-6	bit 0 s-a-1	192	14	[0.300, 0.340]
14-7	bit 1 s-a-0	64	13	[0.320, 0.357]
14-8	bit 1 s-a-1	192	13	[0.300, 0.339]
14-9	bit 2 s-a-0	64	19	[0.314, 0.353]
14-10	bit 2 s-a-1	192	4	[0.280, 0.316]
14-11	bit 3 s-a-0	64	23	[0.301, 0.339]
14-12	bit 3 s-a-1	192	21	[0.285, 0.323]
15-1	ADD -> SUB	148	5	[0.324, 0.363]
15-2	ADD -> AND	255	7	[0.321, 0.358]
15-3	ADD -> OR	148	8	[0.327, 0.366]
15-4	ADD -> EXOR	148	2	[0.333, 0.333]
output of ADD operation stuck-at faults:				
15-5	bit 0 s-a-0	128	4	[0.326, 0.363]
15-6	bit 0 s-a-1	128	10	[0.299, 0.338]
15-7	bit 1 s-a-0	128	2	[0.340, 0.340]
15-8	bit 1 s-a-1	128	2	[0.301, 0.321]
15-9	bit 2 s-a-0	128	4	[0.326, 0.363]
15-10	bit 2 s-a-1	128	8	[0.290, 0.326]
15-11	bit 3 s-a-0	128	13	[0.313, 0.351]
15-12	bit 3 s-a-1	128	9	[0.289, 0.328]

Table 27. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
16-1	OR -> AND	240	2	[0.320, 0.320]
16-2	OR -> EXOR	175	6	[0.320, 0.359]
16-3	OR -> ADD	175	9	[0.320, 0.357]
16-4	OR -> SUB	225	6	[0.305, 0.340]
17-1	AND -> OR	240	3	[0.341, 0.361]
17-2	AND -> EXOR	255	3	[0.336, 0.370]
17-3	AND -> ADD	255	11	[0.305, 0.342]
17-4	AND -> SUB	255	11	[0.313, 0.350]
18-1	SUB -> ADD	256	4	[0.344, 0.381]
18-2	SUB -> AND	256	7	[0.339, 0.377]
18-3	SUB -> OR	256	4	[0.336, 0.370]
18-4	SUB -> EXOR	209	6	[0.350, 0.385]
19-1	SUB -> ADD	224	4	[0.351, 0.388]
19-2	SUB -> AND	255	8	[0.340, 0.375]
19-3	SUB -> OR	225	5	[0.341, 0.371]
19-4	SUB -> EXOR	148	9	[0.319, 0.355]
output of ADD operation stuck-at faults:				
19-5	bit 0 s-a-0	128	2	[0.333, 0.333]
19-6	bit 0 s-a-1	128	10	[0.343, 0.378]
19-7	bit 1 s-a-0	128	3	[0.348, 0.368]
19-8	bit 1 s-a-1	128	6	[0.360, 0.394]
19-9	bit 2 s-a-0	128	4	[0.330, 0.366]
19-10	bit 2 s-a-1	128	11	[0.339, 0.378]
19-11	bit 3 s-a-0	128	5	[0.356, 0.396]
19-12	bit 3 s-a-1	128	10	[0.338, 0.378]
20-1	SUB -> ADD	256	2	[0.340, 0.340]
20-2	SUB -> AND	256	4	[0.318, 0.348]
20-3	SUB -> OR	256	2	[0.346, 0.346]
20-4	SUB -> EXOR	192	5	[0.323, 0.354]

Table 28. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
21-1	AND -> OR	240	7	[0.313, 0.351]
21-2	AND -> EXOR	255	5	[0.317, 0.357]
21-3	AND -> ADD	255	3	[0.333, 0.351]
21-4	AND -> SUB	255	2	[0.353, 0.353]
22-1	ADD -> SUB	148	10	[0.293, 0.329]
22-2	ADD -> AND	240	7	[0.273, 0.313]
22-3	ADD -> OR	175	13	[0.282, 0.320]
22-4	ADD -> EXOR	148	2	[0.320, 0.320]
23-1	AND -> OR	240	16	[0.278, 0.316]
23-2	AND -> EXOR	255	9	[0.279, 0.319]
23-3	AND -> ADD	255	12	[0.283, 0.322]
23-4	AND -> SUB	255	13	[0.288, 0.325]
24-1	ADD -> SUB	224	8	[0.320, 0.357]
24-2	ADD -> AND	255	8	[0.318, 0.353]
24-3	ADD -> OR	175	5	[0.316, 0.356]
24-4	ADD -> EXOR	148	5	[0.308, 0.342]
25-1	ADD -> SUB	148	4	[0.335, 0.368]
25-2	ADD -> AND	255	4	[0.288, 0.328]
25-3	ADD -> OR	175	8	[0.317, 0.355]
25-4	ADD -> EXOR	148	8	[0.309, 0.345]
26-1	OR -> AND	240	8	[0.314, 0.352]
26-2	OR -> EXOR	175	2	[0.326, 0.326]
26-3	OR -> ADD	175	5	[0.306, 0.342]
26-4	OR -> SUB	225	8	[0.301, 0.338]
27-1	AND -> OR	240	5	[0.293, 0.330]
27-2	AND -> EXOR	255	8	[0.302, 0.338]
27-3	AND -> ADD	255	5	[0.316, 0.352]
27-4	AND -> SUB	255	8	[0.310, 0.347]

Table 29. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
28-1	SUB -> ADD	256	5	[0.299, 0.338]
28-2	SUB -> AND	256	5	[0.297, 0.335]
28-3	SUB -> OR	256	4	[0.311, 0.341]
28-4	SUB -> EXOR	192	4	[0.302, 0.330]
29-1	AND -> OR	240	5	[0.300, 0.334]
29-2	AND -> EXOR	255	4	[0.302, 0.337]
29-3	AND -> ADD	255	6	[0.303, 0.341]
29-4	AND -> SUB	255	5	[0.303, 0.336]
30-1	ADD -> SUB	224	13	[0.292, 0.330]
30-2	ADD -> AND	240	18	[0.265, 0.304]
30-3	ADD -> OR	240	15	[0.286, 0.325]
30-4	ADD -> EXOR	224	5	[0.292, 0.328]
31-1	ADD -> SUB	224	10	[0.287, 0.326]
31-2	ADD -> AND	255	7	[0.304, 0.341]
31-3	ADD -> OR	240	12	[0.303, 0.341]
31-4	ADD -> EXOR	224	3	[0.318, 0.338]
32-1	OR -> AND	240	4	[0.332, 0.364]
32-2	OR -> EXOR	175	2	[0.353, 0.353]
32-3	OR -> ADD	175	9	[0.299, 0.337]
32-4	OR -> SUB	225	13	[0.297, 0.335]
33-1	ADD -> SUB	224	4	[0.324, 0.352]
33-2	ADD -> AND	255	2	[0.353, 0.353]
33-3	ADD -> OR	240	4	[0.328, 0.365]
33-4	ADD -> EXOR	224	2	[0.320, 0.320]

Table 30. A Part Results for SN74181 ALU (Continued)

FAULT NO	FAULT	NO. TESTS	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
34-1	OR -> AND	240	10	[0.311, 0.347]
34-2	OR -> EXOR	175	4	[0.325, 0.361]
34-3	OR -> ADD	175	3	[0.335, 0.353]
34-4	OR -> SUB	225	5	[0.326, 0.356]
35-1	SUB -> ADD	256	4	[0.316, 0.344]
35-2	SUB -> AND	256	6	[0.303, 0.334]
35-3	SUB -> OR	256	7	[0.304, 0.338]
35-4	SUB -> EXOR	128	4	[0.309, 0.341]

Table 31. B Part Results for SN74181 ALU

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
11-1	OR -> AND	5	[0.654, 0.690]
14-1	AND -> OR		
15-1	ADD -> SUB		
19-1	SUB -> ADD		

Table 32. C Part Results for SN74181 ALU

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
11-1	OR -> AND	7	[0.831, 0.870]
14-1	AND -> OR		
15-1	ADD -> SUB		
19-1	SUB -> ADD		

Table 33. D Part Results for SN74181 ALU

FAULT NO	FAULT	SAMPLE N TIMES	95% CONFIDENCE COV. INTERVAL
11-1	OR -> AND	4	[0.814, 0.852]
11-2	OR -> EXOR		
11-3	OR -> ADD		
11-4	OR -> SUB		
14-1	AND -> OR		
14-2	AND -> EXOR		
14-3	AND -> ADD		
14-4	AND -> SUB		
15-1	ADD -> SUB		
15-2	ADD -> AND		
15-3	ADD -> OR		
15-4	ADD -> EXOR		
19-1	SUB -> ADD		
19-2	SUB -> AND		
19-3	SUB -> OR		
19-4	SUB -> EXOR		

4.3 *Analysis Summary for Large Combinational Circuits*

In case of SN7483a adder, Table 19 on page 71 and Table 20 on page 72 indicate that all of the micro-operation faults lead to similar 95% confidence coverage intervals. The 95% confidence coverage interval of fault number 1-5 actually corresponds to random testing since all input combinations are tests for that fault. The B part and C part results in Table 21 on page 73 and Table 22 on page 74 show that different micro-operation faults also result in similar accumulated coverage intervals. The results of A, B, C, and D parts support the idea that more tests usually produce higher coverages.

The result tables of SN74181 ALU in previous section also show similar coverage intervals among various perturbations for each micro-operation. This indicates there is no best perturbations for micro-operations.

The experimental results of large combinational circuits 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.

4.4 *Suggestions for Future Work*

1. When the sample circuit gets larger, one may not want to input exhaustive input combinations to both good and faulty modules to get the complete test set for a micro-operation fault under investigation. In this case, the random sampling scheme should be moved from HILO input to GSP2 input. That is, we randomly sample sufficient inputs and then feed them to both good and faulty modules to generate enough tests for calculating 95% confidence coverage interval of a micro-operation fault.

   Note that a test generated from the chip level model of a combinational circuit sometimes consists of a sequence of input combinations which spread on time domain. This is because, in some cases, some conditions need to be set up before the faulty behavior can be propagated to output of the chip level description. Our evaluation methods did not consider such kind of tests for combinational circuits.

2. When dealing with the chip level model of a sequential circuit, a test is usually composed of a sequence of input combinations. Obviously, the longer the sequence is, the higher the fault coverage will be. Since a micro-operation fault may be detected by many tests with various sequence lengths, a measure of effectiveness for micro-operation faults needs to be developed in terms of average coverage and average test sequence length.

5.0 Conclusions

An automatic procedure, which employs a statistical random sampling technique, is presented to search for the best micro-operation fault model in chip level testing using HDLs. 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 in the sense that it is easier to implement by Barclay's algorithm.

For sequential circuits, a new measure of effectiveness for micro-operation faults is needed to evaluate various micro-operation perturbations.

Bibliography

1. P. K. Lala, Fault Tolerant & Fault Testable Hardware Design, Prentice/Hall International, 1985.
2. J. R. Armstrong, Chip Level Modeling With VHDL, to be published.
3. M. A. Breuer and A. D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Woodland Hills, Calif., 1976.
4. J. P. Hayes, "Fault Modeling", IEEE Design and Test of Computers, pp. 88-95, April, 1985.
5. E. I. Muehldorf and A. D. Savkar, "LSI Logic Testing - An Overview", IEEE Trans. on Computers, pp. 1-17, Jan., 1981.
6. J. Roth, "Diagnosis of automata failures: a calculus and a method", IBM J. of R&D, July 1966.
7. M. A. Breuer, A. D. Friedman and A. Iosupovicz, "A survey of the art of design automation", IEEE Computer, pp.58-75, Oct., 1981.
8. Y. Levendel and P. Mellon, "Test generation algorithms for computer hardware description languages", IEEE Trans. on computers, pp.577-588, July, 1982.
9. J. R. Armstrong, "Chip-Level Modeling with HDLs", IEEE Design & Test of Computers, pp.8-18, Feb. 1988.
10. J. R. Armstrong, A. K. Gupta, and J. Stewart, "Functional Fault modeling for VLSI Devices", Final Report for IBM contract YD 190121, 1984.
11. A. K. Gupta, "Functional Fault Modeling and Test Vector Development for VLSI Systems", Master's Thesis, Virginia Polytechnic Institute and State University, March 1985.
12. D. S. Barclay, "An Automatic Test Generation Algorithm for Chip-Level Circuit Descriptions", Master's Thesis, Virginia Polytechnic Institute and State University, January 1986.
13. D. S. Barclay and J. R. Armstrong, "A Heuristic Chip-Level Test Generation Algorithm", 23rd Design Automation Conference, pp. 257-262, June 1986.
14. VHDL Language Reference Manual, Version 7.2, IR-MD-045-2, August 1985.
15. VHDL Language Reference Manual, IEEE Preliminary Version, Oct., 1986.

16. GSP User's Manual, E.E. Dept., Virginia Polytechnic Institute and State University, December 1982.

17. J. R. Armstrong and D. E. Devlin, "GSP: A Simulator for LSI", 18th Design Automation Conference, pp.518-524, 1981.

18. A. K. Gupta and J. R. Armstrong, "Functional Fault Modeling and Simulation for VLSI devices", 22nd Design Automation Conference, pp.720-726, 1985.

19. C. H. Cho, J. R. Armstrong and F. G. Gray, "Chip Level Fault Modeling and Test Generation", E.E. Dept., Virginia Polytechnic Institute and State University, August, 1986.

20. 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.

21. P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits", IEEE Trans. on Computer, pp. 215-222, March 1981.

22. J. J. Thomas, "Automated Diagnostic Test Programs for Digital Networks", Computer Design, pp. 63-67, August 1971.

23. S. Su and T. Lin, "Functional testing techniques for digital LSI/VLSI systems", 21th Design Automation Conference, pp. 517-528, 1984.

24. S. Thatte and J. Abraham, "Test generation for microprocessors", IEEE Trans. on Computers, pp. 429-441, June 1980.

25. K. Lai, "Functional testing of digital systems", Ph.D. Thesis, Comp. Sci. Dept., Carnegie-Mellon Univ., December 1981.

26. M. A. Breuer and A. D. Friedman, "Functional level primitives in test generation", IEEE Trans. on Computers, pp.223-235, March 1980.

27. T. Lin and S. Su, "The S-Algorithm : A Promising Solution for Systematic Functional Test Generation", IEEE Trans. on Computer-Aided Design, pp. 250-263, July 1985.

28. T. Lin and S. Su, "VLSI Functional Test Pattern Generation - A Design and Implementation", 1985, International Test Conference, pp. 922-929, 1985.

29. G. W. Snedecor and W. G. Cochran, "Statistical Methods", The Iowa State University Press, 1980.

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

31. HILO-3 User Manual, GenRad Inc., June 1985.

Appendix A. GSP2 Programs for SN7483A Adder Experiments

A.1 The CONNECT Program

```
!*****************************************************
* This is the GSP2 connect file for SN7483a adder
* experiment system. This system is used to obtain
* a test set for an injected fault.
*****************************************************!

CONNECT

clock_gen : clock[0|1] OUTPUT,
counter   : clock[0|1] INPUT;

clock_gen : clock[0|1] OUTPUT,
comparator : clock[0|1] INPUT;

counter   : excitation_vector[0|9,0|4] OUTPUT,
clag      : a[0|4] INPUT,
claf      : a[0|4] INPUT;

counter   : excitation_vector[0|9,4|4] OUTPUT,
clag      : b[0|4] INPUT,
claf      : b[0|4] INPUT;

counter   : excitation_vector[0|9,8|1] OUTPUT,
clag      : carry_in[0|1] INPUT,
claf      : carry_in[0|1] INPUT;

clag      : output_vector[0|4] OUTPUT,
comparator: good_output[0|4] INPUT;

clag      : carry_out[0|1] OUTPUT,
comparator: carry_g[0|1] INPUT;

claf      : output_vector[0|4] OUTPUT,
comparator: bad_output[0|4] INPUT;

claf      : carry_out[0|1] OUTPUT,
comparator: carry_f[0|1] INPUT

ENDCONN
```

A.2 The CLOCK Module Program

```
!************************************************
* This GSP2 module produces clock which is used
* to trigger the counter module and the
* comparator module.
************************************************!

MODULE clock_gen

LITERALS

  hi = '#B1';
  lo = '#B0'

PINS
  start[0|1] : INPUT;
  clock[0|1] : OUTPUT

DECLARE

  count : INTEGER

EVENT clock_enable ON @RISE( start );

  LOOP count := 1 TO 515 BY 1 DO

    BEGIN
      clock := hi PROP 5;
      clock := lo PROP 10;
      WAIT 10
    END

  ENDLOOP

ENDMOD
```

A.3 The COUNTER Module Program

```
!*****************************************************************
* This is a 9 bit binary counter. It counts from
* 0 0000 0000 to 1 1111 1111.
* First 4 bits feed a0 through a3 lines of the CLAG and CLAF.
* Next  4 bits feed b0 through b3 lines of the CLAG and CLAF.
* The MSB of this counter feeds carry inputs of CLAG, CLAF.
*****************************************************************!

MODULE counter

LITERALS
            hi = '#B1';
            lo = '#B0'

PINS
        clock [0|1], clear [0|1] : INPUT;
        excitation_vector [0|9]  : OUTPUT

DECLARE

        count[0|10] : REGISTER VALUE 0

EVENT run ON @RISE ( clock );

    BEGIN

      IF clear = hi THEN
        BEGIN
            count := @VECT(0,10) PROP 1;
            wait 2;
            excitation_vector := count[0|9] PROP 1
        END
      ELSE
        IF @INT(count) = 511 THEN
            BEGIN
                count := @VECT(0,10) PROP 1;
                wait 2;
                excitation_vector := count[0|9] PROP 1
            END
         ELSE
            BEGIN
                count := @VECT(@INT(count) + 1,10) PROP 1;
                wait 2;
                excitation_vector := count[0|9] PROP 1
            END
         ENDIF
      ENDIF
 END

ENDMOD
```

A.4 The Good Module Program for the SN7483a Adder

```
!*********************************************
* This is the good GSP2 module for the 4-bit
* carry-look-ahead adder ( SN7483a ).
*********************************************!

MODULE clag

PINS
        a[0|4] : INPUT;
        b[0|4] : INPUT;
        carry_in[0|1] : INPUT;
        output_vector[0|4] : OUTPUT ;
        carry_out[0|1] : OUTPUT

DECLARE

        carry_vector[0|5] : REGISTER ;
        int_result1[0|5]  : REGISTER ;
        int_result2[0|5]  : REGISTER ;
        extended_bit[0|1] : REGISTER VALUE 0

EVENT sum ON @CHANGE(@CONCAT(a,b,carry_in));

BEGIN

      IF carry_in = #B1
        THEN
            carry_vector := #B00001
        ELSE
            carry_vector := #B00000
      ENDIF;

      !*********************************************************
      *The numbers listed on the right of a statement indicate
      *the micro-operation numbers.
      *********************************************************!

      int_result1 := @ADD(@CONCAT(extended_bit,a),@CONCAT(extended_bit,b));
                                                  !----1----!
      int_result2 := @ADD(int_result1, carry_vector); !----2----!

      IF int_result2[4|1] =#B1
        THEN
            carry_out := #B1 PROP 4
        ELSE
            carry_out := #B0 PROP 4
      ENDIF;

      output_vector := int_result2[0|4] PROP 4

END
ENDMOD
```

A.5 The Faulty Module Program for the SN7483a Adder

```
!************************************************
* This is the faulty GSP2 module for the 4-bit
* carry-look-ahead adder ( SN74181 ).
* The injected micro-operation fault is fault
* number 1-1 : ADD failed to SUB.
************************************************!

MODULE claf

PINS
        a[0|4] : INPUT;
        b[0|4] : INPUT;
        carry_in[0|1] : INPUT;
        output_vector[0|4] : OUTPUT ;
        carry_out[0|1] : OUTPUT

DECLARE

        carry_vector[0|5] : REGISTER ;
        int_result1[0|5]  : REGISTER ;
        int_result2[0|5]  : REGISTER ;
        extended_bit[0|1] : REGISTER VALUE 0

EVENT sum ON @CHANGE(@CONCAT(a,b,carry_in));

BEGIN

      IF carry_in = #B1
        THEN
            carry_vector := #B00001
        ELSE
            carry_vector := #B00000
      ENDIF;

      !-- micro-operation substitution fault 1-1-- ADD failed to SUB--!

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

      int_result2 := @ADD(int_result1, carry_vector);

      IF int_result2[4|1] =#B1
        THEN
            carry_out := #B1 PROP 4
        ELSE
            carry_out := #B0 PROP 4
      ENDIF;

      output_vector := int_result2[0|4]  PROP  4

END

ENDMOD
```

A.6 The COMP Module Program

```
!*****************************************************
* This GSP2 module compares the outputs of the good
* module and the faulty module.
*****************************************************!

MODULE  comparator


LITERALS

     gate_delay = '1'

PINS

     clock [0|1] : INPUT ;
     carry_g [0|1] : INPUT ;
     carry_f [0|1] : INPUT ;
     good_output [0|4] : INPUT ;
     bad_output [0|4] : INPUT ;
     comp_output [0|6] : OUTPUT

DECLARE

     sign_bit    [0|1] : REGISTER VALUE 0 ;
     comp_result [0|6] : REGISTER VALUE 0 ;
     good_vector [0|6] : REGISTER VALUE 0 ;
      flt_vector [0|6] : REGISTER VALUE 0

EVENT compare ON @RISE(clock);

     BEGIN

        WAIT 8;
        good_vector := @CONCAT(sign_bit,good_output,carry_g);
         flt_vector := @CONCAT(sign_bit,bad_output,carry_f);
        comp_result := @SUB ( flt_vector, good_vector ) ;
        comp_output := comp_result PROP gate_delay

     END

ENDMOD
```

A.7 The GSP2 Command Language File

```
!*********************************************
* This is the GSP2 Command Language ( GCL )
* file for monitoring simulation of the
* SN7483a adder experiment system.
*********************************************!

set timelimit = 5130
set obase = D
set module = clock_gen
dep start  = B1 at 5
set module = counter
dep clear  = B1 at 9
dep clear  = B0 at 11
trace variable count
set module = comparator
trace variable comp_output
```

Appendix B. GSP2 Programs for SN74181 ALU Experiments

B.1 The CONNECT Program

```
!*****************************************************
* This is the GSP2 connect file for the SN74181 ALU
* experiment system. The system is used to obtain
* a test set for a injected fault.
*****************************************************!

CONNECT

clock_gen  : clock[0|1] OUTPUT,
counter    : clock[0|1] INPUT;

clock_gen  : clock[0|1] OUTPUT,
comparator : clock[0|1] INPUT;

counter    : excitation_vector[0|9,0|4] OUTPUT,
alug       : a[0|4] INPUT,
aluf       : a[0|4] INPUT;

counter    : excitation_vector[0|9,4|4] OUTPUT,
alug       : b[0|4] INPUT,
aluf       : b[0|4] INPUT;


counter    : excitation_vector[0|9,8|1] OUTPUT,
alug       : ncn[0|1] INPUT,
aluf       : ncn[0|1] INPUT;

alug       : f[0|4] OUTPUT,
comparator : good_output[0|8,0|4] INPUT;

alug       : aeqb[0|1] OUTPUT,
comparator : good_output[0|8,4|1] INPUT;

alug       : g[0|1] OUTPUT,
comparator : good_output[0|8,5|1] INPUT;

alug       : ncn4[0|1] OUTPUT,
comparator : good_output[0|8,6|1] INPUT;

alug       : p[0|1] OUTPUT,
comparator : good_output[0|8,7|1] INPUT;

aluf       : f[0|4] OUTPUT,
comparator : bad_output[0|8,0|4] INPUT;

aluf       : aeqb[0|1] OUTPUT,
comparator : bad_output[0|8,4|1] INPUT;

aluf       : g[0|1] OUTPUT,
comparator : bad_output[0|8,5|1] INPUT;
```

```
aluf       : ncn4[0|1] OUTPUT,
comparator : bad_output[0|8,6|1] INPUT;

aluf       : p[0|1] OUTPUT,
comparator : bad_output[0|8,7|1] INPUT

ENDCONN
```

B.2 The CLOCK Module Program

```
!************************************************
* This GSP2 module produces clock which is used
* to trigger the counter module and the
* comparator module.
************************************************!

MODULE clock_gen

LITERALS

  hi = '#B1';
  lo = '#B0'

PINS

  start[0|1] : INPUT;
  clock[0|1] : OUTPUT

DECLARE

  count : INTEGER

EVENT clock_enable ON @RISE( start );

  LOOP count := 1 TO 515 BY 1 DO

    BEGIN
      clock := hi PROP 5;
      clock := lo PROP 10;
      WAIT 10
    END

  ENDLOOP

ENDMOD
```

B.3 The COUNTER Module Program

```
!****************************************************************
* This is a 9-bit binary counter. It counts from
* 0 0000 0000 to 0 1111 1111
* First 4 bits feed a0 through a3 lines of the ALUG and ALUF.
* Next 4 bits feed b0 through b3 lines of the ALUG and ALUF.
* The MSB of this counter feeds ncn inputs of ALUG, ALUF.
* The m and s values are controlled in GSP2 Command
* Language file ( GCL ) to sensitize particular program
* statement.
****************************************************************!

MODULE counter

LITERALS
            hi = '#B1';
            lo = '#B0'
PINS
      clock [0|1], clear [0|1] : INPUT;
      excitation_vector  [0|9] : OUTPUT

DECLARE
      count[0|10] : REGISTER VALUE 0

EVENT run ON @RISE ( clock );

    BEGIN

      IF clear = hi THEN
        BEGIN
            count := @VECT(0,10) PROP 1;
            wait 2;
            excitation_vector := count[0|9] PROP 1
        END
      ELSE
        IF @INT(count) = 511 THEN
            BEGIN
                count := @VECT(0,10) PROP 1;
                wait 2;
                excitation_vector := count[0|9] PROP 1
            END
         ELSE
            BEGIN
                count := @VECT(@INT(count) + 1,10) PROP 1;
                wait 2;
                excitation_vector := count[0|9] PROP 1
            END
        ENDIF
      ENDIF
  END
ENDMOD
```

B.4 The Good Module Program for the SN74181 ALU

```
!*****************************************************
* This is the good GSP2 module for the SN74181 ALU.
*****************************************************!

MODULE alug

LITERALS

 Selection Code definitions

f0 = '#B0000'; f1 = '#B0001'; f2 = '#B0010'; f3 = '#B0011'; f4 = '#B0100';
f5 = '#B0101'; f6 = '#B0110'; f7 = '#B0111'; f8 = '#B1000'; f9 = '#B1001';
f10= '#B1010'; f11= '#B1011'; f12= '#B1100'; f13= '#B1101'; f14= '#B1110';
f15= '#B1111';

 User Flags

zero = '#B0000';
one =  '#B0001'

PINS

s[0|4], m[0|1], ncn[0|1], a[0|4], b[0|4]: INPUT;

f[0|4], aeqb[0|1], g[0|1], ncn4[0|1], p[0|1]: OUTPUT

!** Main Program **!

EVENT start ON @CHANGE(@CONCAT(s[0|4],m[0|1],ncn[0|1],b[0|4],a[0|4]));

BEGIN

g := @NOT(@OR(@AND(@NOT(@OR(@AND(@NOT(b[3|1]),s[1|1]),
                @AND(s[0|1],b[3|1]),
                a[3|1]))),
        @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
           @NOT(@OR(@AND(@NOT(b[2|1]),s[1|1]),
                @AND(s[0|1],b[2|1]),
                a[2|1]))),
        @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
           @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
                @AND(a[2|1],s[2|1],@NOT(b[2|1])))),
           @NOT(@OR(@AND(@NOT(b[1|1]),s[1|1]),
                @AND(s[0|1],b[1|1]),
                a[1|1]))),
        @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
           @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
                @AND(a[2|1],s[2|1],@NOT(b[2|1])))),
           @NOT(@OR(@AND(b[1|1],s[3|1],a[1|1]),
```

```
                @AND(a[1|1],s[2|1],@NOT(b[1|1]))),
            @NOT(@OR(@AND(@NOT(b[0|1]),s[1|1]),
                @AND(s[0|1],b[0|1]),
                a[0|1])))) PROP 1;

p := @NOT(@AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
            @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
                @AND(a[2|1],s[2|1],@NOT(b[2|1])))),
            @NOT(@OR(@AND(b[1|1],s[3|1],a[1|1]),
                @AND(a[1|1],s[2|1],@NOT(b[1|1])))),
            @NOT(@OR(@AND(b[0|1],s[3|1],a[0|1]),
                @AND(a[0|1],s[2|1],@NOT(b[0|1])))))) PROP 1;

IF m = #B1

THEN

   !** mode = 1 , logic functions **!

   CASE s[0|4] OF

   !************************************************************
   * The numbers listed on the right of a statement indicate
   * the micro-operation numbers.
   ************************************************************!

    f0: f[0|4] := @NOT(a) PROP 1;

    f1: f := @NOT(@OR(a,b)) PROP 1;          !-----1-----!

    f2: f := @AND(@NOT(a),b) PROP 1;         !-----2-----!

    f3: f := zero PROP 1;

    f4: f := @NOT(@AND(a,b)) PROP 1;         !-----3-----!

    f5: f := @NOT(b) PROP 1;

    f6: f := @EXOR(a,b) PROP 1;              !-----4-----!

    f7: f := @AND(a,@NOT(b)) PROP 1;         !-----5-----!

    f8: f := @OR(@NOT(a),b) PROP 1;          !-----6-----!

    f9: f := @NOT(@EXOR(a,b)) PROP 1;        !-----7-----!

    f10: f := b PROP 1;

    f11: f := @AND(a,b) PROP 1;              !-----8-----!

    f12: f := one PROP 1;

    f13: f := @OR(a,@NOT(b)) PROP 1;         !-----9-----!

    f14: f := @OR(a,b) PROP 1;               !----10-----!
```

```
    f15: f := a PROP 1
  ENDCASE
ELSE
  !** mode = 0 , arithmetic operations **!
  IF ncn = #B1
    THEN
      !** no carry **!
     CASE s[0|4] OF
      f0: f := a PROP 1;
      f1: f := @OR(a,b) PROP 1;              !----11-----!
      f2: f := @OR(a,@NOT(b)) PROP 1;        !----12-----!
      f3: f := #B1111 PROP 1;
      f4: f := @ADD(a,@AND(a,@NOT(b))) PROP 1; !--13,14--!
      f5: f := @ADD(@OR(a,b),@AND(a,@NOT(b))) PROP 1; !-15,16,17-!
      f6: f := @SUB(@SUB(a,b),one) PROP 1;           !--18,19---!
      f7: f := @SUB(@AND(a,@NOT(b)),one) PROP 1;     !--20,21---!
      f8: f := @ADD(a,@AND(a,b)) PROP 1;             !--22,23---!
      f9: f := @ADD(a,b) PROP 1;                     !-----24---!
     f10: f := @ADD(@OR(a,@NOT(b)),@AND(a,b)) PROP 1; !-25,26,27-!
     f11: f := @SUB(@AND(a,b),one) PROP 1;           !--28,29---!
     f12: f := @ADD(a,a) PROP 1;                     !-----30---!
     f13: f := @ADD(@OR(a,b),a) PROP 1;              !--31,32---!
     f14: f := @ADD(@OR(a,@NOT(b)),a) PROP 1;        !--33,34---!
     f15: f := @SUB(a,one) PROP 1                    !-----35---!
    ENDCASE
  ELSE
    !** with carry **!
    CASE s[0|4] OF
```

```
      f0: f := @ADD(a,one) PROP 1;
      f1: f := @ADD(@OR(a,b),one) PROP 1;
      f2: f := @ADD(@OR(a,@NOT(b)),one) PROP 1;
      f3: f := zero PROP 1;
      f4: f := @ADD(@ADD(a,@AND(a,@NOT(b))),one) PROP 1;
      f5: f := @ADD(@ADD(@OR(a,b),@AND(a,@NOT(b))),one) PROP 1;
      f6: f := @SUB(a,b) PROP 1;
      f7: f := @AND(a,@NOT(b)) PROP 1;
      f8: f := @ADD(@ADD(a,@AND(a,b)),one) PROP 1;
      f9: f := @ADD(@ADD(a,b),one) PROP 1;
     f10: f := @ADD(@ADD(@OR(a,@NOT(b)),@AND(a,b)),one) PROP 1;
     f11: f := @AND(a,b) PROP 1;
     f12: f := @ADD(@ADD(a,a),one) PROP 1;
     f13: f := @ADD(@ADD(@OR(a,b),a),one) PROP 1;
     f14: f := @ADD(@ADD(@OR(a,@NOT(b)),a),one) PROP 1;
     f15: f := a PROP 1
    ENDCASE
  ENDIF
ENDIF;

wait 2;
aeqb := @AND(f[0|1],f[1|1],f[2|1],f[3|1]) PROP 1;
ncn4 := @NOT(@AND(g,@NOT(p),ncn)) PROP 1

END

ENDMOD
```

B.5 The Faulty Module Program for the SN74181 ALU

```
!*******************************************************
* This is the faulty GSP2 module for the SN74181 ALU.
* The injected micro-operation fault is fault number
* 1-1 : OR failed to AND.
*******************************************************!

MODULE aluf

LITERALS

 Selection Code definitions

f0 = '#B0000'; f1 = '#B0001'; f2 = '#B0010'; f3 = '#B0011'; f4 = '#B0100';
f5 = '#B0101'; f6 = '#B0110'; f7 = '#B0111'; f8 = '#B1000'; f9 = '#B1001';
f10= '#B1010'; f11= '#B1011'; f12= '#B1100'; f13= '#B1101'; f14= '#B1110';
f15= '#B1111';

 User Flags

zero = '#B0000';
one =  '#B0001'

PINS

s[0|4], m[0|1], ncn[0|1], a[0|4], b[0|4]: INPUT;

f[0|4], aeqb[0|1], g[0|1], ncn4[0|1], p[0|1]: OUTPUT

!** Main Program **!

EVENT start ON @CHANGE(@CONCAT(s[0|4],m[0|1],ncn[0|1],b[0|4],a[0|4]));

BEGIN

g := @NOT(@OR(@AND(@NOT(@OR(@AND(@NOT(b[3|1]),s[1|1]),
                @AND(s[0|1],b[3|1]),
                a[3|1]))),
          @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
             @NOT(@OR(@AND(@NOT(b[2|1]),s[1|1]),
                @AND(s[0|1],b[2|1]),
                a[2|1]))),
          @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
             @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
                @AND(a[2|1],s[2|1],@NOT(b[2|1])))),
             @NOT(@OR(@AND(@NOT(b[1|1]),s[1|1]),
                @AND(s[0|1],b[1|1]),
                a[1|1]))),
          @AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1])))),
             @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
```

```
                @AND(a[2|1],s[2|1],@NOT(b[2|1]))),
            @NOT(@OR(@AND(b[1|1],s[3|1],a[1|1]),
                @AND(a[1|1],s[2|1],@NOT(b[1|1]))),
            @NOT(@OR(@AND(@NOT(b[0|1]),s[1|1]),
                @AND(s[0|1],b[0|1]),
                a[0|1])))) PROP 1;

p := @NOT(@AND(@NOT(@OR(@AND(b[3|1],s[3|1],a[3|1]),
                @AND(a[3|1],s[2|1],@NOT(b[3|1]))),
            @NOT(@OR(@AND(b[2|1],s[3|1],a[2|1]),
                @AND(a[2|1],s[2|1],@NOT(b[2|1]))),
            @NOT(@OR(@AND(b[1|1],s[3|1],a[1|1]),
                @AND(a[1|1],s[2|1],@NOT(b[1|1]))),
            @NOT(@OR(@AND(b[0|1],s[3|1],a[0|1]),
                @AND(a[0|1],s[2|1],@NOT(b[0|1])))))) PROP 1;

IF m = #B1

THEN

   mode = 1 , logic functions

  CASE s[0|4] OF

   f0: f[0|4] := @NOT(a) PROP 1;

   f1: f := @NOT(@AND(a,b)) PROP 1; !-- fault number 1-1 : OR failed to AND --!

   f2: f := @AND(@NOT(a),b) PROP 1;

   f3: f := zero PROP 1;

   f4: f := @NOT(@AND(a,b)) PROP 1;

   f5: f := @NOT(b) PROP 1;

   f6: f := @EXOR(a,b) PROP 1;

   f7: f := @AND(a,@NOT(b)) PROP 1;

   f8: f := @OR(@NOT(a),b) PROP 1;

   f9: f := @NOT(@EXOR(a,b)) PROP 1;

   f10: f := b PROP 1;

   f11: f := @AND(a,b) PROP 1;

   f12: f := one PROP 1;

   f13: f := @OR(a,@NOT(b)) PROP 1;

   f14: f := @OR(a,b) PROP 1;

   f15: f := a PROP 1
```

```
  ENDCASE
ELSE
  !** mode = 0 , arithmetic operations **!
  IF ncn = #B1
    THEN
      !** no carry **!
      CASE s[0|4] OF
        f0: f := a PROP 1;
        f1: f := @OR(a,b) PROP 1;
        f2: f := @OR(a,@NOT(b)) PROP 1;
        f3: f := #B1111 PROP 1;
        f4: f := @ADD(a,@AND(a,@NOT(b))) PROP 1;
        f5: f := @ADD(@OR(a,b),@AND(a,@NOT(b))) PROP 1;
        f6: f := @SUB(@SUB(a,b),one) PROP 1;
        f7: f := @SUB(@AND(a,@NOT(b)),one) PROP 1;
        f8: f := @ADD(a,@AND(a,b)) PROP 1;
        f9: f := @ADD(a,b) PROP 1;
       f10: f := @ADD(@OR(a,@NOT(b)),@AND(a,b)) PROP 1;
       f11: f := @SUB(@AND(a,b),one) PROP 1;
       f12: f := @ADD(a,a) PROP 1;
       f13: f := @ADD(@OR(a,b),a) PROP 1;
       f14: f := @ADD(@OR(a,@NOT(b)),a) PROP 1;
       f15: f := @SUB(a,one) PROP 1
      ENDCASE
    ELSE
      !** with carry **!
      CASE s[0|4] OF
        f0: f := @ADD(a,one) PROP 1;
```

```
      f1: f := @ADD(@OR(a,b),one) PROP 1;
      f2: f := @ADD(@OR(a,@NOT(b)),one) PROP 1;
      f3: f := zero PROP 1;
      f4: f := @ADD(@ADD(a,@AND(a,@NOT(b))),one) PROP 1;
      f5: f := @ADD(@ADD(@OR(a,b),@AND(a,@NOT(b))),one) PROP 1;
      f6: f := @SUB(a,b) PROP 1;
      f7: f := @AND(a,@NOT(b)) PROP 1;
      f8: f := @ADD(@ADD(a,@AND(a,b)),one) PROP 1;
      f9: f := @ADD(@ADD(a,b),one) PROP 1;
     f10: f := @ADD(@ADD(@OR(a,@NOT(b)),@AND(a,b)),one) PROP 1;
     f11: f := @AND(a,b) PROP 1;
     f12: f := @ADD(@ADD(a,a),one) PROP 1;
     f13: f := @ADD(@ADD(@OR(a,b),a),one) PROP 1;
     f14: f := @ADD(@ADD(@OR(a,@NOT(b)),a),one) PROP 1;
     f15: f := a PROP 1
    ENDCASE
  ENDIF
ENDIF;

wait 2;
aeqb := @AND(f[0|1],f[1|1],f[2|1],f[3|1]) PROP 1;
ncn4 := @NOT(@AND(g,@NOT(p),ncn)) PROP 1

END

ENDMOD
```

B.6 The COMP Module Program

```
!*****************************************************
* This GSP2 module compares the outputs of the good
* module and the faulty module.
*****************************************************!

MODULE  comparator


LITERALS

     gate_delay = '1'

PINS

     clock [0|1] : INPUT ;
     good_output [0|8] : INPUT ;
     bad_output [0|8]  : INPUT ;
     comp_output [0|9] : OUTPUT

DECLARE

     sign_bit    [0|1] : REGISTER VALUE 0 ;
     comp_result [0|9] : REGISTER VALUE 0 ;
     good_vector [0|9] : REGISTER VALUE 0 ;
      flt_vector [0|9] : REGISTER VALUE 0

EVENT compare ON @RISE(clock);

     BEGIN

        WAIT 8;
        good_vector :=  @CONCAT(sign_bit,good_output);
         flt_vector :=  @CONCAT(sign_bit,bad_output);
        comp_result :=  @SUB ( flt_vector, good_vector ) ;
        comp_output :=  comp_result PROP gate_delay

     END

ENDMOD
```

B.7 The GSP2 Command Language File

```
!*********************************************
* This is the GSP2 Command Language ( GCL )
* file for monitoring simulation of the
* SN74181 ALU experiment system.
* This GCL file sensitizes fault # 1
* statement by selecting m = 1 and s = 0001
*********************************************!

set timelimit = 2570
set obase = D
set module = clock_gen
dep start  = B1 at 5
set module = counter
dep clear  = B1 at 9
dep clear  = B0 at 11

set module = alug
dep m      = B1 at 9
dep s      = B0001 at 9

set module = aluf
dep m      = B1 at 9
dep s      = B0001 at 9

set module = counter
trace variable count
set module = comparator
trace variable comp_output
```

Appendix C. Programs for Calculating the Intervals

C.1 The SCRIPT Module Program

```
#************************************************
# This is the SCRIPT module program, a HP-UX
# Bourne Shell script. It is the master of the
# experiment system used to calculate the 95%
# confidence interval.
#************************************************

echo 'This is the master program SCRIPT !'

# Initialize the COV and DEMO files. #

rm cov
rm demo

# Ask for initial n value and initial random seed. #

echo Enter a value for n
read p1
echo Enter seed value
read p2
rm nseed
echo > nseed $p1
echo > > nseed $p2

rm result

# Repeat the sampling process until the RESULT file is created. #

until test -s result

do

# Execute the WRITE.P module program. #

write.e not_test nseed clawave

# Perform the HILO fault simulation. #

hilo < hilo.cmds

# Extract the coverage value from the HILO simulation output. #

cat print.hilo | while read p1 p2 p3 p4 p5
 do
  if [ $p1 ] ;then
   if [ $p1 = "FAULT" ] ; then
    if [ $p2 = "COVER" ] ; then
     if [ $p3 = "=" ] ; then
      echo > > cov $p4
      break
     fi
    fi
```

```
  fi
 fi
done

#*************************************************************
# Change the form of coverages to a form can be input
# to WRITE.P module. For example, change 95.5% to 95.5 .
#*************************************************************

ex cov < ex.cmds

# Execute the INT.P module program. #

int.e cov ttable result

done
```

C.2 The EXTRACT Program

```
#************************************************************
# This is the EXTRACT script which extracts a not-test set
# from a GSP2 log file.
#************************************************************

cat cla11.log | while read p1 p2 p3 p4 p5 p6 p7 p8 p9
do
if [ $p1 ] ; then
 if [ $p1 = "Time" ] ; then
  if [ $p4 = "COUNTER:COUNT" ] ; then

# set $1 = $p9 #

    set -- $p9
  else

# $p4 = "COMPARATOR:COMP_OUTPUT" #

    if [ $p9 = "0" ] ; then
      echo > > not_test11 $1
    fi
  fi
 fi
fi
done
```

C.3 The WRITE.P Module Program

```
{************************************************************
* This is the WRITE.P module program.
* This program first inputs not-test set of a fault and
* convert it to test set. Second, it receives the initial
* random seed and the initial sample size n, which is zero,
* from the NSEED module. It then write the new random seed
* and new sample size n + 1 back to the NSEED module for
* next sampling use. Finally, it picks a test from the test
* set according to the random number generated and writes
* the HILO waveform file for the test.
************************************************************}

program writewave(input, output, not_test, nseed, clawave);

type
    integer_string  = array[1..1000] of integer;
    t_string        = array[0..1000] of integer;
    vector_type     = array[1..10] of char;

var
    i, j, k, m, n, seed       : integer;
    nt, ran                   : integer_string;
    t                         : t_string;
    test_vector               : vector_type;
    not_test, nseed, clawave  : text;

{******************************************************
* This function inputs the original seed and the test
* set size m and then generates the new seed for next
* sampling use and the new random number for selecting
* a test from a test set.
******************************************************}

function random(var seed, m : integer) : integer;

const
    modulus = 65536;
    multiplier = 25173;
    increment  = 13849;

begin{random}
    seed := ((multiplier * seed) + increment) mod modulus;
    random := trunc((seed / modulus) * 1000) mod m;
end;{random}


{************************************************************
* This procedure converts the randomly sampled test number
* t[ran[n]] to the corresponding test pattern.
************************************************************}

procedure convert(var p : integer;
```

```
            var t_vector : vector_type);

var
    count : integer;

begin{convert}
    for count := 10 downto 1 do
    begin
        if odd(p) then
          t_vector[count] := '1'
        else
          t_vector[count] := '0';
        p := p div 2;
    end;
end;{convert}



begin{main}

    append(output, 'demo');
    writeln;
    writeln;
    writeln('********** Here is a new round ! --- SN7483a CLA **********');

    {*** read in the not_test set for a micro-operation fault ***}

    reset(not_test);
    i := 0;
    while not eof(not_test) do
        begin
            i := i + 1;
            readln(not_test,nt[i]);
        end;

    writeln;
    writeln('number of total not-tests = ',i:3);

    {****** convert the not_test set to the test set ************}

    k := 1;
    m := 0;
    for j := 0 to 511 do
        if j < > nt[k] then
          begin
              t[m] := j;
              m := m + 1;
          end
        else
          k := k + 1;

    writeln('number of total tests = ', m:3);

    {**** read in the sample size n and the random seed *******}

    reset(nseed);
```

```
    readln(nseed,n);
    readln(nseed,seed);

    {** generate a random number and write back new n,seed to NSEED **}

    n := n + 1;
    ran[n] := random(seed,m);

    rewrite(nseed);
    writeln(nseed, n);
    writeln(nseed, seed);

    writeln('n = ', n:3);
    writeln('random test number ran[', n:3,'] = ', ran[n]:3);
    writeln('the random test t[ran[n]] = ', t[ran[n]]:3);

    {** convert the randomly sampled test number to test pattern **}

    convert(t[ran[n]], test_vector);

    {** write the HILO waveform for the test pattern **}

    rewrite(clawave);
    writeln(clawave, 'waveform clawave');
    writeln(clawave, 'stimulus c0,b4,b3,b2,b1,a4,a3,a2,a1 = 0;');
    writeln(clawave, 'response c4,s4,s3,s2,s1 = x;');
    writeln(clawave, '10 ',' a4 = ', test_vector[7],
                     ' a3 = ', test_vector[8],
                     ' a2 = ', test_vector[9],
                     ' a1 = ', test_vector[10]);
    writeln(clawave, '  ',' b4 = ', test_vector[3],
                     ' b3 = ', test_vector[4],
                     ' b2 = ', test_vector[5],
                     ' b1 = ', test_vector[6],
                     ' c0 = ', test_vector[2],';');
    writeln(clawave, '90  strobe;');
    writeln(clawave, '100  finish.');

end.{main}
```

C.4 The HILO.CMDS Program

```
#*****************************************
# This the hilo.cmds file. It first inputs
# the waveform file from HP-UX environment
# to HILO environment. Then, it performs
# the HILO fault simulation.
#*****************************************

            # ctrl M ---- carriage return #
            # ctrl M ---- carriage return #
            # ctrl M ---- carriage return #
rf clawave
simulate sn7483a clawave
faultsim
stuck * .
quit
```

C.5 The EX.CMDS Program

```
#*********************************************
# This is the ex.cmds script. It first locates
# the symbol % , deletes % and then exits.
#*********************************************

/%
s/%//
x
```

C.6 The INT.P Module Program

```
{*******************************************************
* This is the INT.P module program.
* This program inputs coverages from previous random
* samplings and decides whether the 95% confidence
* interval has been achieved. If the interval has
* been achieved, the RESULT file is created;
* otherwise, the RESULT is not created and the
* sampling procedure is repeated again.
*******************************************************}

program interval(input,output,cov,ttable,result);

type
    real_string = array[1..1000] of real;
    set_list    = set of 1..150;

var
    i,j,n,df      : integer;
    c,t           : real_string;
    cb,s,temp1,temp2,d,upper,lower : real;
    cov,ttable,result  : text;
    t_set       : set_list;

begin{main}
     append(output, 'demo');

     {*** read in previous sampling coverages values ***}

     n := 0;
     reset(cov);
     while not eof(cov) do
          begin
               n := n + 1;
               readln(cov,c[n]);
               c[n] := c[n] * 0.01;
               writeln('coverage c[',n:3,'] =', c[n]:5:3);
          end;

      writeln('n = ',n:3);

     if n > 1 then
      begin

          {************************************************
          * calculate the arithmetic mean of coverages (cb)
          * and the sample standard deviation (s).
          ************************************************}

          temp1 := 0;
          temp2 := 0;

          for i := 1 to n do
```

```
      temp1 := temp1 + c[i];
cb := temp1 / n;

for i := 1 to n do
      temp2 := temp2 + SQR(c[i] - cb);
s := SQRT(temp2/(n-1));

{*** read in the TTABLE t values ***}


t_set := [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
        19,20,21,22,23,24,25,26,27,28,29,30,35,40,45,
        50,55,60,70,80,90,100,120,150];

j := 0;
reset(ttable);
for j := 1 to 150 do
   if j in t_set then
        readln(ttable, t[j]);

df := n - 1;
if df in t_set then
  begin

       {*** calculate the interval distance ***}

       d := t[df] * s/SQRT(n);

       if d < = 0.02 then

         {* The interval resolution is good enough. *}

         begin

           {** calculate the upper and lower bounds **}

           upper := cb + d;
           lower := cb - d;

           {** create the RESULT file **}

           rewrite(result);
           writeln(result, '95% confidence interval has been achieved!');
           writeln(result,'n = ', n:3, ' the interval : [',lower:5:3,',',upper:5:3,']');
           writeln;
           writeln('95% confidence interval has been achieved!');
           writeln('n= ', n:3, ', the interval :[',lower:5:3,',',upper:5:3,']');
         end
       else

         {* The interval resolution is not good enough. *}

         writeln(' d > 0.02, need more test !');
  end
else
```

```
            {*** The t[df] is not listed in the ttable. ***}
            writeln('t[',df:3,'] is not in ttable, need more tests !');
        end
      else
            {* Need at least two samplings to calculate the interval. *}
            writeln(' n = 1, need more tests !');
end.{main}
```

C.7 The TTABLE Module t Data

```
12.706
4.303
3.182
2.776
2.571
2.447
2.365
2.306
2.262
2.228
2.201
2.179
2.160
2.145
2.131
2.120
2.110
2.101
2.093
2.086
2.080
2.074
2.069
2.064
2.060
2.056
2.052
2.048
2.045
2.042
2.030
2.021
2.014
2.008
2.004
2.000
1.994
1.989
1.986
1.982
1.980
1.960
```

The vita has been removed from the scanned document