National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

TEST GENERATION FOR DETECTING MULTIPLE STUCK FAULTS IN SYNCHRONOUS SEQUENTIAL CIRCUITS USING BOOLEAN DIFFERENCE AND TRANSITION MATRIX TECHNIQUES

by

Thiep V. Nguyen, B. A. Sc.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

ISBN 0-315-82533-2

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor, Dr. Sunil R. Das, for his encouragement and guidance throughout the course of this research.

Appreciation is also expressed to the staff, professors and graduate students of the Department of Electrical Engineering for their helps and efforts in providing the academic environment.

I also wish to thank all members of my family, especially my wife, Trang, for their love, supports, and patience. Finally, I would like to dedicate this thesis to my children, for the time being: daughter Mai, and the second little one whom we are expecting in about two weeks.

ABSTRACT

The Boolean difference is a mathematical concept which has proved its usefulness in the study of single and multiple stuck-at faults in combinational circuits. This tool of analysis was extended to cover multiple stuck-at faults in synchronous sequential circuits as well. In this dissertation, modifications to previous work are presented, together with the development of a new method for deriving the required shortest test sequence to detect a specified multiple fault. First, the vector Boolean difference technique is utilized to determine the input vector that will produce a difference in output between the fault-free and faulty circuits with both starting in the same initial state. If that detection cannot be achieved immediately, then the state transition matrices of both circuits are combined and used to form a matrix of detecting state pairs. Each of these pairs comprises of the present states of both circuits for which an output difference will be detected by an input vector. The detecting tree is then built leading the two circuits from the same initial state to the first detecting state found to complete the search for the shortest test sequence. Besides being able to identify, at an early stage, faults that are undetectable, this algorithm guarantees the generation of a shortest test sequence, if one exists, for every multiple stuck-at fault in a synchronous sequential circuit having a synchronizing sequence or a known initial state. A computer program was also written as a tool to automatically generate test sequences for detecting single or multiple faults in both combinational and synchronous sequential circuits.

Chapter I
INTRODUCTION

The development of computers has been stimulated so greatly by integrated-circuit technology that, in order to keep pace with this rapid advance, the reliability and fault detection in digital circuits have emerged as an important principal research area in fault-tolerant computing. As the complexity and popularity of today's VLSI realization of digital circuits increase, the testing problems involving fault detection, fault analysis, and test generation become more difficult, and so the testing cost contributes to an increasingly larger proportion of the total product cost. This difficulty can be reduced greatly by the development of faster and more efficient algorithms for test-pattern generation besides the use of design techniques to enhance testability. In this dissertation, we will present methods for test generation to detect multiple stuck-at faults in synchronous sequential circuits.

This thesis is organized as follows:

Chapter 1 is devoted to providing an introduction to the testing problem, and an overview of related test generation techniques from which the new method presented in this dissertation finds its adaptation to the current fault tolerant research area. First, the testing method and the fault model used for this new algorithm are discussed. Since test generation techniques for combinational circuits form the basis for the development of

most of those for sequential circuits, they are necessarily reviewed, especially the D-Algorithm and the Boolean difference method. An overview of test generation methods for sequential circuits then follows with some concentration in the Extended D-algorithm and the Nine-Value Algorithm, in which the element of time is taken into account. Finally, the new method presented here is briefly described.

Chapter 2 describes all the notations used for the multiple fault and the sequential circuit under test. The types of input vectors and the definitions of tests considered in this dissertation are then given. The fault equivalence concept for reducing the number of faults under consideration is also discussed.

Chapter 3 shows how the Boolean difference is applied to the case of multiple faults in synchronous sequential circuits as a first attempt to find the single test vectors. Examples for different cases are also given.

Chapters 4 and 5 describe the technique using detection matrix and detecting tree, respectively. These are used to find the shortest test sequence of length greater than one, when the application of Boolean difference in Chapter 3 indicates that no single test vector exists.

Applications and experimental results from using a computer program for both Moore and Mealy synchronous sequential circuits can be found in Chapter 6.

Chapter 7 discusses and compares the present method with other methods. Then the conclusions are stated.

Appendix A describes the method for finding Boolean difference using the Karnaugh map or any such representation. This forms the basis for writing the C program which is shown in Appendix B for implementing the algorithm proposed in this dissertation. Appendix C shows the program output listings for examples of testing a Moore circuit (type '164 IC) and a Mealy circuit (type '97 IC).

1.1 Method of Testing

Logic circuits are tested by applying a sequence of patterns or vectors at the primary inputs. A test for a fault is an input vector or sequence of vectors that will produce different primary outputs when compared with the correct (expected) ones of the fault-free circuit. This is equivalent to saying that, a test is the same input vector or sequence of vectors which are applied, at the same time, to both fault-free and faulty circuits which start in the same initial state, until there is a difference in the primary outputs between the two circuits. In a combinational circuit, a specific stuck fault can be tested by a single input vector. Testing a fault in sequential circuit, in general, requires a sequence of vectors.

1.2 Fault Modeling

During chip fabrication, many types of defects can occur; for example, break in single lines, lines shorted to ground, excessive delays, etc. In general, the effect of a fault is represented by means of a model. The most commonly modeled faults are line stuck type faults which are considered in this work. In this model, we assume that any one line in the circuit may have a fault such that the signal on this line is fixed to either a logic 1 (stuck-at-1), or a logic 0 (stuck-at-0) irrespective of the input vector. Main advantages of stuck fault model are:

- They have proven to be very effective in representing the fault behavior of actual devices and in measure of test quality.
- They are modeled at the logic level and are independent of technology.
- They can be analyzed by known methods.

For a circuit with a total of m lines, if there exists only one fault at a time in a circuit, there are at most 2m possible single faults. If there exist m faults at the same time, the total number of possible multiple faults increase dramatically to $3^m - 1$, since any line may be fault-free, stuck at 0 (s-a-0), or stuck at 1 (s-a-1). For this reason, the single fault assumption is frequently used, and also since studies have shown that single fault detection test set can detect almost all multiple faults [8]. However, faults which occur during circuit manufacture frequently affect several parts of the circuit and consequently can be more closely modeled as multiple stuck faults rather than as single stuck faults [7]. In this dissertation, multiple fault in synchronous sequential circuit is considered in order to show the completeness in test generation of the new algorithm presented herein.

1.3 Test Generation for Combinational Circuits - Overview

Since early 1960s, numerous algorithms have been proposed for generating test vectors for combinational and sequential circuits. Some of these are effective and widely used in practice; others are of limited practical interest. Most approaches are structural or topological, i.e. they construct the test-input vectors by analyzing the circuit topology.

1.3.1 Path Sensitization Method

Most well-known test generation algorithms for combinational circuits make use of the path sensitization idea in one form or another. Of these, the oldest and undoubtedly the best known is the D-algorithm.

1.3.1.1 D-Algorithm

In the D-algorithm [34], a five-value {0, 1, X (unknown), D, $\bar{D}$} calculus is utilized to carry out the sensitization and justification in a formal manner. The symbols D and its complement $\bar{D}$ represent fault effects. If a signal has a value 1 in the fault-free circuit and 0 in the faulty circuit, its value is denoted by D. The complementary situation is denoted by $\bar{D}$.

The D-algorithm consists of three parts, namely, forward implication, D-drive, and backward justification or consistency check. In forward implication, that is similar to

logic simulation, output values of logic gates are determined for given input values. In D-drive, the fault effect (D and $\bar{D}$) is propagated toward primary outputs. An exhaustive decision tree is traversed while assigning binary (0 and 1) values to internal nodes of the circuit in specific order. Backward justifications are performed iteratively traversing another decision tree to justify node values from primary inputs. If any signal value implies a conflict, an alternative value is assigned to the node by performing a backup in the tree traversal. This process is repeated until the effect of the fault reaches a primary output at which point the values on primary inputs correspond to a test.

The D-algorithm propagates fault effects along multiple paths which is essential to guarantee a test, if such a test exists. However, it has been pointed out that this algorithm is inefficient in generating tests for circuits with many Exclusive-OR gates. The degradation in performance arises due to excessive amount of backtracking.

1.3.1.2 PODEM, FAN and CONT

The backtracking problem in the D-algorithm is improved by a test generation algorithm called Path Oriented Decision Making (PODEM) [19]. In this method a branch and bound technique is used to make PODEM implementations running an order of magnitude faster than the D-algorithm in most circuits. This is achieved by assigning values only to the primary inputs which are then propagated towards internal lines by the implication, and thus backtracking can occur only at the primary inputs.

In [17] a technique is described to further accelerate a path-sensitization algorithm like PODEM called the fanout-oriented test-generation (FAN) which does extensive anal-

ysis of the circuit connectivity in a preprocessing step to minimizing backtracking. It performs special processing of fanout points and has been shown to be more efficient and faster than PODEM.

Another recent algorithm called the concurrent test generation (CONT) [37] minimizes backtracking by a different approach. If the target fault is found to be undetectable by the current input vector, CONT tries to switch the target to another undetected fault. FAN and CONT can produce a reduction factor of 2-5 in computing time over PODEM [4].

Test generation can be automatic. A prerequisite for automatic test generation is an algorithm that can be programmed. Such algorithms mostly work on the principle of path sensitization. Efficient programs for combinational circuits are available based on algorithms just mentioned such as D-algorithm, PODEM, and FAN.

1.3.2 Boolean Difference Method

Besides those commonly known structural methods of path sensitization listed above, there have been some algebraic methods [35] reported which generate test patterns by manipulating algebraic formulas. One of these, the Boolean difference method is a mathematical approach which captures the basic concepts of path sensitization in algebraic terms. Let F(X) be the output function of the fault-free circuit and F_π (X) the output function of the circuit with the presence of a given fault π . The set $\{ t \mid F(X) \oplus F_\pi (X) = 1 \}$, as defined later, is the set of all input vectors that distinguish between the two functions.

The algebraic methods are quite elegant and complete , i.e. claiming to produce a test whenever one exists. However, in general, it has been a difficult task to manipulate algebraic equations to derive test for a given fault. They have the disadvantage of requiring a large quantity of time and memory, which may make them impractical for large circuits.

1.4 Test Generation for Sequential Circuits - Overview

Test generation for sequential circuits remains to be a challenge in spite of a history of attempts dating back to the late 1960s. In particular, it has been recognized as the most difficult problem in the area of fault detection. The difficulty comes from the existence of memory elements. With memory elements, such as latches or flip-flops in a circuit, the outputs depend not only on the current inputs but also on the operation history (stored internal states). Of course, it is possible to facilitate sequential circuit testing by adding some extra hardware, which then enhances the controllability and observability of the circuit [39]. However, the test hardware increases hardware overhead and can degrade circuit performance. Thus, before using valuable chip space, test generation without adding extra hardware should be considered.

In the sequential circuit, since the stored internal states can retain their values over time, combinational test generation methods can thus be applied to sequential circuit if the element of time is introduced. There are few sequential test generation methods that are of practical significance. In general, there are three different approaches considered as discussed below [8].

1.4.1 Iterative Array Approach

In this approach a combinational model for a sequential circuit is constructed by regenerating the feedback signal from previous-time copies of the circuit. Thus the timing behavior of the circuit is approximated by combinational levels. Topological analysis algorithms that activate faults and sensitize paths through these multiple copies of the iterative combinational circuit are then used to generate tests.

A synchronous sequential circuit M can be generally modeled as in Figure 1.1. By cutting the feedback loops where the clocked flip-flops are, the iterative combinational circuit M^p can be formed as in Figure 1.2 [16]. The combinational circuits C_i , where i=0,..,p, are all identical to the combinational portion C of the original sequential circuit M, and each of them corresponds to the time frame i. The inputs of C_i include the primary input I(i) and the pseudo-input SI(i) obtained from the feedback signals produced by C_{i-1}. Similarly, O(i) and SO(i-1) are the primary output and pseudo-output, respectively of the time frame i. In this transformation, the clocked flip-flops of M are modeled as combinational elements F_i, where i=0,..,p. These are referred to as pseudo- flip-flops. Figure 1.3 represents the pseudo- flip-flops corresponding to a D, a T, and an SR flip-flops.

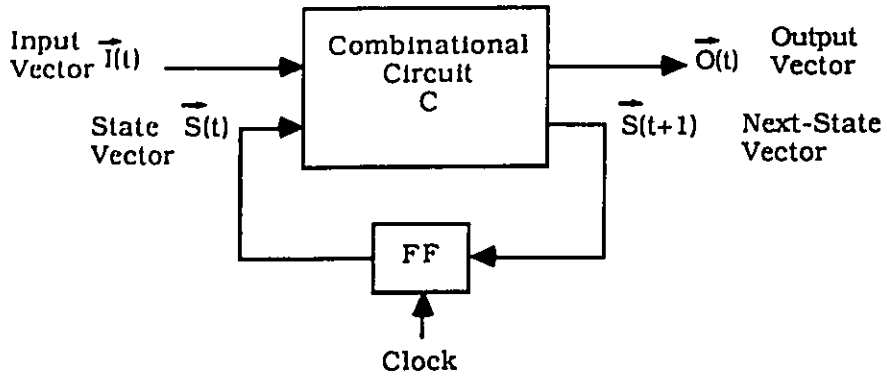


Figure 1.1 Synchronous Sequential Circuit M

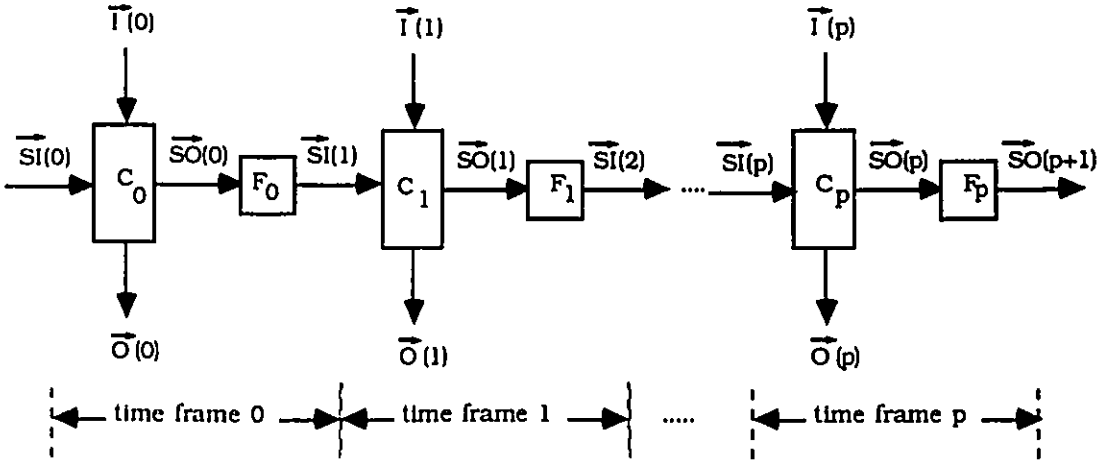


Figure 1.2 Corresponding Iterative Combinational Circuit M^p

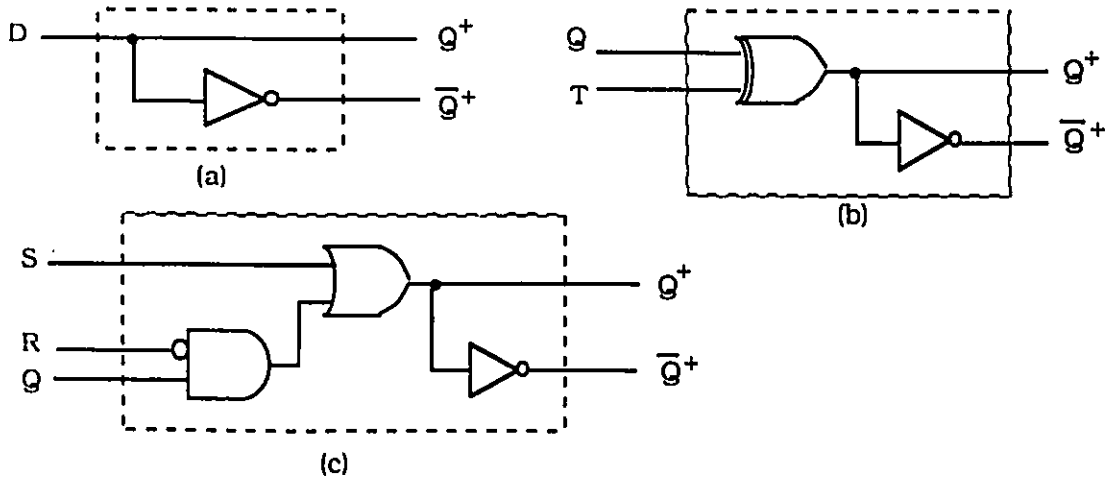


Figure 1.3 (a) D , (b) T and (c) SR Pseudo Flip-flop

1.4.1.1 Extended D-algorithm

In the extended D-algorithm [25, 33], in order to find a test sequence to detect a single stuck-at fault, the number of time frames needed in the iterative combinational model must be determined. The five-valued D-algorithm is then applied to the current time frame, say jth copy. If the D-algorithm assigns a value to any pseudo-input, this assignment is then justified in the (j-1)th copy. Similarly, if the fault effect is propagated to a pseudo-output, this propagation must continue into the (j+1)th and subsequent copies until the fault effect reaches a primary output. Although the D-algorithm guarantees to generate a test for a fault in combinational circuit, if one exists, for sequential circuits, the extended D-algorithm does not guarantee a test sequence for a fault even if one exists.

1.4.1.2 Nine-Value Algorithm

In the Nine-Value algorithm [31], the five-value algebra used in the D-algorithm is extended to a nine-value one which consists of (1/1, 1/0, 1/X, 0/1, 0/0, 0/X, X/1, X/0, and X/X) representing ordered pairs of states of the fault-free and faulty circuits. The five values used in the D-algorithm are actually a subset of these nine values; a 1 used in D-algorithm is equivalent to 1/1, a 0 is 0/0, and X is X/X, a D is 1/0, and the $\bar{D}$ is 0/1. The additional partially specified values, 0/X, 1/X, X/1 and X/0, provide a greater degree of freedom in test generation. The Nine-Value algorithm takes into account the possible repeated effects of the fault in a sequential circuit and so a test can be guaranteed, if one exists, for every stuck-at fault in a synchronous sequential circuit having a synchronizing sequence (an input sequence which when applied to a sequential circuit results in a unique final state independent of the initial state). However, the implementation of this algorithm is very complex and may be inefficient for large circuit.

1.4.1.3 SOFTG, EBT, and BACK

Another method called the Simulator-Oriented Fault Test Generator (SOFTG) [36] is primarily a sequential version of PODEM. Here, tests are constructed by tracing backward through the circuit in much the same way as a combinational test generator, but all forward signal propagations are carried out by an event-driven simulator.

All sequential test generation algorithms mentioned above have a process flow that is bidirectional in time since the starting event is the fault activation at the site of the fault, and in general, the detection will take place at a future time while the primary inputs must assume their values in the past, where time refers to the time of events in the circuit. A unidirectional path sensitization algorithm called the Extended Backtrace (EBT) [28, 29, 30] works backward both spatially and in time from a selected output toward the fault, where a path and an output pin with best observability must be selected for a given fault.

While the EBT algorithm sensitizes a preselected path, the BACK algorithm [10] only preselects a primary output to which it sensitizes all paths starting at the fault site.

1.4.2 Verification-Based Approach

The verification-based approach relies on determining whether or not the circuit under test is operating in accordance with its state table. Sequential Circuit Test Search System (SCIRTSS) [22] applies the D-algorithm to the combinational portion of the circuit to obtain a test vector which is then split between primary inputs and the present state. At this point, through the state graph, suitable paths are searched to find an input sequence to bring the circuit to this present state and another sequence to propagate the stored fault effect to a primary output. In order to limit the effort, the searches for these paths are only conducted over a restricted state diagram and a small portion of the much larger set of data states.

In another method, a test generation for finite state machines [27] applies PODEM with iterative-array type of processing for forward propagation. The backward justification phase is replaced by a procedure that attempts to search for a path in the state transition graph (STG) from a given reset state to the present state required by PODEM. Again, in order to minimize the complexity of the STG, a partial state diagram is constructed containing all valid states of the finite machine but only a few transition edges.

1.4.3 Functional Approach

A functional approach [6] uses a high-level description of the circuit to generate test sequences that will verify whether the designed functions are being performed correctly. This approach often uses restricted fault models like the line stuck faults on inputs and

outputs of functional blocks or higher-level fault models such as errors in the truth table of a combinational block, or a change in the state table of a sequential functional block. The functional testing is often the method for testing very large circuits like microprocessors. However, it is difficult to evaluate the quality of functional test vectors. In many cases, these tests may not be capable of detecting every possible failure that can occur.

1.5 New Algorithm Proposed in The Dissertation

Test generation algorithms for combinational and sequential circuits are briefly reviewed above to show some basic ideas from which a new algorithm presented in this dissertation has evolved. This new algorithm can be considered as a combination of the iterative-array and the verification-based approaches mentioned in Sections 1.4.1 and 1.4.2: It applies Boolean difference algorithm to an iterative combinational model, but only one copy is needed; and if necessary, it then makes use of the state table to build special matrices for searching for the shortest test sequence, if one exists.

The Boolean difference method is a well-known mathematical concept which has found significant application in the single fault analysis of combinational logic circuits. Several authors have also extended this technique to the multiple fault case in combinational circuits [15, 24], and in synchronous sequential circuits [20, 21]. In this dissertation, the analysis discussed in [20, 21] is reviewed and modified, together with a new method developed to find the shortest test-input sequence.

Assume that both the fault-free and faulty circuits start in the same known initial state. First, treating the present and next state of the iterative combinational circuit as pseudo-input and pseudo-output vector respectively, and with the present state set to the known initial state, a vector Boolean difference technique is applied to the primary output vector to determine the single input vector that will produce a difference in at least one of the primary outputs between the fault-free and faulty circuits. This difference may not be achieved immediately, i.e. the fault cannot be detected yet by the single input vectors, because of one of the following two reasons:

- Case 1: All the primary outputs do not depend on the fault at all, no matter what the present state the circuit under test (CUT) is in.
- Case 2: All the primary outputs do not depend on the fault only when the CUT is in the known initial state. This is a subset of case 1.

For case 1, the Boolean difference is then applied to the pseudo-output or next-state functions, without setting the present state to the known initial state, to check if there is any difference in at least one of the next-state functions between the two circuits. If there is none, we can then conclude that the specified fault is undetectable or redundant. It should be noted here that, this is the earliest checkpoint where undetectable fault is identified.

For case 2, or for the situation in which the next-state difference is found for case 1, it is promising that a test sequence of length greater than one may be found. The state transition matrices of the fault-free and faulty circuits are then combined and used to form the matrix of detecting state pairs, or shortly detection matrix. Each pair of states in

the detection matrix corresponds to the present states of the fault-free and faulty circuits in which an output difference will be obtained by a single input vector. It is obvious that if the detection matrix is empty, i.e. no detecting pair exists, then the fault is said to be undetectable at this second early checkpoint. If there is at least one detecting pair in the detection matrix, the detecting tree is then built, and based on the matrices, the path leading the two circuits from the initial state pair to the first detecting pair is searched for. If the detecting tree terminates or has reached its predefined maximum level before reaching the first detecting pair, then the fault is said to be undetectable.

One of the primary attributes of this new algorithm is its completeness: It guarantees the generation of the shortest test sequence, if one exists, for every multiple stuck-at fault in a synchronous sequential circuit having a synchronizing sequence or a known initial state. Since the Boolean difference can be found by using the Karnaugh map and since the matrix representation is so straightforward in computing technique, a computer program was also developed as a basic tool for an automatic test generator (ATG) to generate test sequence for detecting single or multiple stuck-at faults in both combinational and synchronous sequential circuits. Experimental applications to both Moore and Mealy sequential circuits are also presented.

Chapter II

CIRCUIT AND FAULT DESCRIPTIONS

2.1 Description of Circuit Under Test (CUT)

Consider a synchronous sequential circuit with m inputs, n outputs, and b bits of memory as shown in Figure 1.1. In vector notation, the circuit can be described as :

Input vector = $\vec{I}(t) = [I_1(t), I_2(t), \ldots, I_m(t)]$

Output vector = $\vec{O}(t) = [O_1(t), O_2(t), \ldots, O_n(t)]$

State vector = $\vec{S}(t) = [S_1(t), S_2(t), \ldots, S_b(t)]$

where t is the time parameter, and $I_i(t)$, $1 \le i \le m$, $O_j(t)$, $1 \le j \le n$, and $S_k(t)$, $1 \le k \le b$ are Boolean variables.

The behavior of the circuit can be described by two vector-based Boolean functions :

$$\vec{O}(t) = G[\vec{I}(t), \vec{S}(t)], \quad t \ge 0 \qquad (1)$$

$$\vec{S}(t+1) = F[\vec{I}(t), \vec{S}(t)], \quad t \ge 0 \qquad (2)$$

where $\vec{O}(t)$ and $\vec{S}(t+1)$ are the output and next-state vectors, respectively, and G and F are the output and next-state functions, respectively.

Since line stuck type faults are being considered here, the set of interested lines should now be described. In a given sequential circuit, each line which can be faulty is labeled with a unique number. The numbered line of interest is any connection between :

- a primary input and an input of a gate, or
- a primary input and a fanout point, or
- a fanout point and an input of a gate, or
- a primary output and an output of a gate, or
- a primary output and a fanout point, or
- an output of a gate and an input of another gate.

Note that these line numbers must be consistent with the corresponding lines when being transformed from the original synchronous sequential circuit to the iterative combinational circuit in which no other new lines are numbered.

A line numbered x has a logical value L_x which is dependent on the input and present-state vectors. Let p faulty lines, x1, x2,..., xp, be a subset of the total numbered lines. If these p lines involve either function G or F, then the faulty-line vector $\vec{L}(t)$ has the form:

$$\vec{L}(t) = [\, L_{x1}(t), L_{x2}(t), ..., L_{xp}(t) \,]$$

where $L_{x1}, L_{x2},..., L_{xp}$ are the logical values of lines x1, x2,..., xp, respectively.

Since the internal lines are also dependent on the input and present-state vectors, the output and next-state Equations (1) and (2) can then be modified to explicitly indicate their dependence upon logical values of the faulty lines as follows :

$$\vec{O}(t) = G[\vec{I}(t), \vec{S}(t), \vec{L}(t)], \quad t \geq 0 \qquad (3)$$

$$\vec{S}(t+1) = F[\vec{I}(t), \vec{S}(t), \vec{L}(t)], \quad t \geq 0 \qquad (4)$$

Assume that the initial state vector of the circuit is a known $\vec{S}(0) = SI$. It is obvious that when the circuit is in the initial state at time t=0, the output vector in Equation (3) can be replaced by $\vec{S}(0)$ or SI, and Equation (5) is obtained as :

$$\vec{O}(0) = G[\vec{I}(0), SI, \vec{L}(0)] \qquad (5)$$

Let $G_j[\vec{I}(t), \vec{S}(t), \vec{L}(t)]$ be the output function corresponding to the jth output variable, $O_j(t)$. Similarly, let $F_k[\vec{I}(t), \vec{S}(t), \vec{L}(t)]$ be the next-state function corresponding to the kth variable of the next state, $S_k(t+1)$. They can be denoted as:

$$O_j = \vec{O}_j(t) = G_j[\vec{I}(t), \vec{S}(t), \vec{L}(t)], \quad 1 \leq j \leq n \qquad (6)$$

$$S^+_k = \vec{S}_k(t+1) = F_k[\vec{I}(t), \vec{S}(t), \vec{L}(t)], \quad 1 \leq k \leq b \qquad (7)$$

2.2 Multiple Fault Description

A multiple fault is representable as the logical values of a group of lines being permanently stuck at the logical value one (s-a-1) or zero (s-a-0). So, a multiple fault involving p faulty lines is the case in which a specific subset of lines x1, x2, ..., xp are being stuck-at $a_{x1}, a_{x2}, ..., a_{xp}$, respectively, and are represented by L_{x1} (s-a-a_{x1}), L_{x2} (s-a-a_{x2}),..., L_{xp}(s-a-a_{xp}), respectively, where $a_{xi} \in \{0,1\}$, i=1,2,...,p. The next-state and output functions (3) and (4), resulting from setting $L_{x1}(t) = a_{x1}, L_{x2}(t) = a_{x2}, ..., L_{xp}(t) = a_{xp}$, are $F[\vec{I}(t), \vec{S}(t), a_{x1}, a_{x2},..., a_{xp}]$ and $G[\vec{I}(t), \vec{S}(t), a_{x1}, a_{x2},..., a_{xp}]$, respectively.

2.2.1 Input Vector Types

In an iterative combinational circuit derived from a synchronous sequential circuit with some multiple fault, say, L_{x1} (s-a-a_{x1}), ..., L_{xp} (s-a-a_{xp}), an input vector applied at the primary and pseudo-inputs is classified as one of the following three types : Type I vector attempts to drive the logical value of each of the faulty lines, L_{x1}, ..., L_{xp}, to the values, $\bar{a}_{x1}$, ..., $\bar{a}_{xp}$, respectively. Type II vector attempts to drive only a subset, while type III attempts to drive none, of the faulty lines to logical values which are the complements of the faulty values. For example, for double fault L_i (s-a-1), L_j (s-a-0), type I vectors attempt to drive (L_i, L_j) with (0,1), while type II with (0,0) or (1,1), and type III with (1,0).

It should be noted here that type I or type II vector is essential in an attempt to excite the fault, and all three types of vectors may be needed to propagate the fault effect to the primary outputs. Since, in general, a test for a multiple fault in a sequential circuit may require a sequence of input vectors, this sequence may consist of all of the three types of vectors, with at least one type I or type II vector being present in it. Note that in a combinational circuit, type III vector has no meaning, and so not mentioned in [15, 20, 21, 24], since test vectors are single type I or type II vectors which must excite the fault and, at the same time, cause the fault effect to appear at the primary outputs.

2.2.2 Test Definitions for Multiple Faults

We assume that both the fault-free and faulty circuits start in the same known initial state $\vec{S}(0) = SI$. This assumption may not always hold.

Definition 1 : A test, t_{SI}, for a multiple fault in a sequential circuit starting in initial state SI is defined as a finite sequence of input vectors, which consists of all three types of vectors with at least one type I or type II vector being present in it, and of which the application will cause the output vector of the faulty circuit to differ from that of the fault-free circuit.

In the following discussion, if not specified, a test means the one defined in Definition 1.

Definition 2 : A minimal test is the shortest sequence of input vectors which will determine the presence of the fault by the very first difference in at least one of the primary outputs between the faulty and fault-free circuits.

Only minimal test sequence is of interest in the present approach, since it is necessary and sufficient to conclude that the specified multiple fault is detected, or the circuit under test (CUT) is faulty.

Definition 3 : A minimal test sequence is defined as a Type I or Type II Test if it consists of a single type I or type II vector, respectively.

Since a fault in a combinational circuit can only be tested by single input vectors, it is obvious that the complete test set for a fault in a combinational circuit consists of two disjoint subsets of type I and type II tests [24]. A test for a fault in a sequential circuit may be a single input vector or a sequence of vectors. Single vector test or type I or II test for sequential circuit is described in Chapter 3 when Boolean difference technique is applied to the primary output vector.

In considering multiple faults, we must determine reduced faults and discuss about equivalent faults.

2.2.3 Equivalent Faults and Reduced Faults

Strong equivalence of faults in synchronous sequential circuit is identical to the concept of equivalence of faults in the iterative combinational circuits derived from the synchronous sequential circuits by breaking all feedback loops, and considering the current or present state vector as a pseudo-input vector and the next-state vector as a pseudo-output vector.

Definition 4 : Two faults α and β in a sequential circuit are said to be strongly equivalent if their effect is indistinguishable, i.e. the output and next-state functions of the circuit with fault α are identical to those of the circuit with fault β.

If a set of faults are equivalent, any test which detects one of them will detect all of them and no test will distinguish them. Hence, in test generation it is only necessary to

explicitly consider one fault from each set of equivalent faults. Therefore, from a set of equivalent multiple faults, the one that consists of the least number of single faults should be selected for test generation purposes to minimize the testing time and resources.

Definition 5 : For any two equivalent multiple faults α and β, α is said to be a *reduced multiple fault* if its set of single faults is a subset of the set of single faults of β.

Only reduced multiple faults are considered in this research.

Let us consider the sequential circuit given in Figure 2.1. After breaking the feedback loops and using the T pseudo-flip-flop model shown in Figure 1.3(b), the iterative combinational circuit is obtained in Figure 2.2. Note that the line numbers in Figure 2.2 are consistent with those in Figure 2.1, and that a fault can still only be detected by the difference in the primary outputs (O_1, O_2) between the fault-free and faulty circuits. The next state lines (S_1^+, S_2^+) are just conventional and are needed in the process of propagating the fault effect to an output difference later if that cannot be achieved immediately by the single input vectors.

In Figure 2.2, for example, the triple fault L_5 (s-a-0), L_9 (s-a-1), L_{14} (s-a-1) is equivalent to the single fault L_{14} (s-a-1) which was also a subset of the triple one. Therefore, it is necessary to test only for the presence of the reduced fault L_{14} (s-a-1).

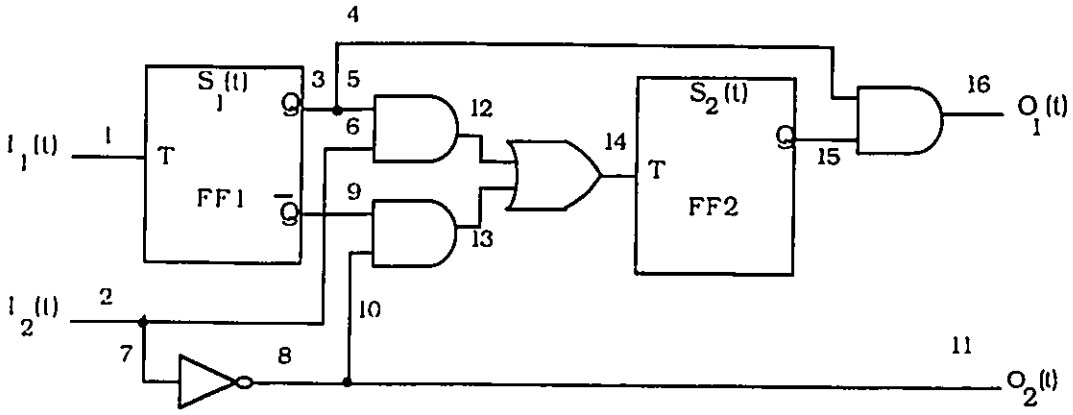


Figure 2.1 Synchronous Sequential Circuit

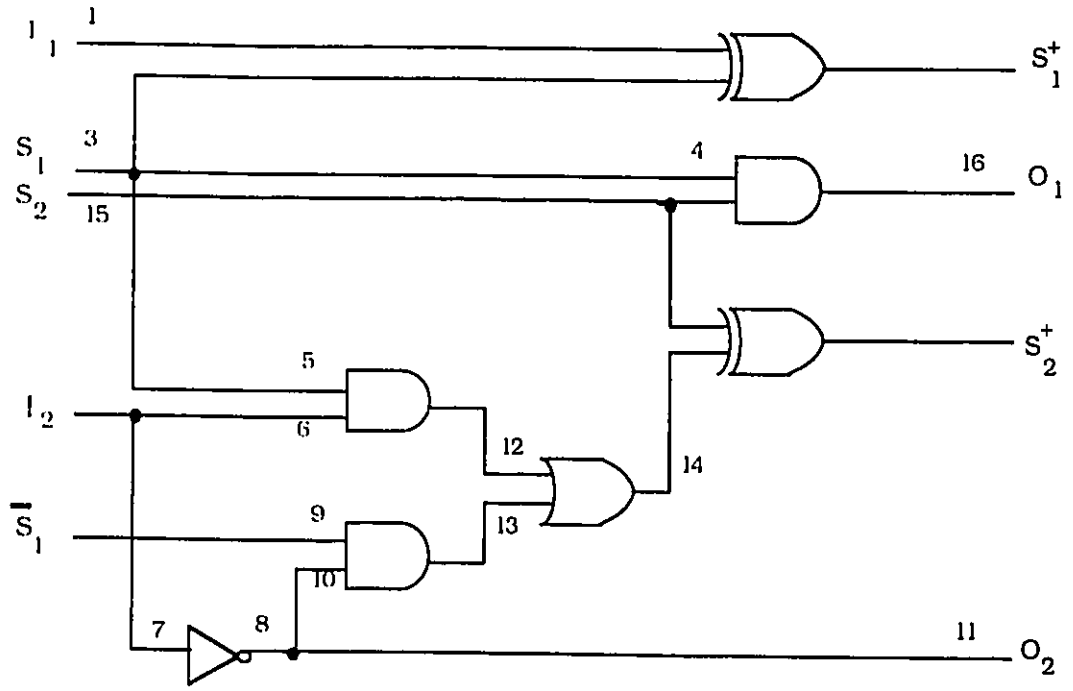


Figure 2.2 Iterative Combinational Circuit for Figure 2.1

Chapter III

BOOLEAN DIFFERENCE EXPRESSIONS FOR MULTIPLE FAULTS IN SYNCHRONOUS SEQUENTIAL CIRCUITS

First, the basic concepts of Boolean difference for faults in combinational circuits are briefly reviewed.

Let $\vec{I}$ be the input vector and L_k be the logical value of the faulty line k. The Boolean difference of a Boolean function $Y(\vec{I},L_k) = Y(L_k)$, with respect to the single variable L_k is defined as [24]:

$$\frac{dY(L_k)}{dL_k} = Y(L_k) \oplus Y(\bar{L}_k) = Y(1) \oplus Y(0) \qquad (8)$$

and is denoted by $\Omega_k Y$.

Similarly, the double Boolean difference of $Y(\vec{I},L_i,L_j) = Y(L_i,L_j)$, with respect to L_i and L_j is defined as [24]:

$$\frac{d^2 Y(L_i,L_j)}{dL_i\, dL_j} = \frac{d}{dL_i}\left(\frac{dY(L_i,L_j)}{dL_j}\right) = \frac{d}{dL_j}\left(\frac{dY(L_i,L_j)}{dL_i}\right)$$

$$= Y(\bar{L}_i \bar{L}_j) \oplus Y(\bar{L}_i L_j) \oplus Y(L_i \bar{L}_j) \oplus Y(L_i L_j)$$

$$= Y(0,0) \oplus Y(0,1) \oplus Y(1,0) \oplus Y(1,1)$$

then the Boolean difference of Y with respect to any subset of 2 variables, L_i and L_j, is defined as:

$$\Omega_{i,j} Y = \frac{d^2 Y(L_i,L_j)}{d L_i \, d L_j} \oplus \frac{dY(L_i,L_j)}{dL_j} \oplus \frac{dY(L_i,L_j)}{dL_i} \qquad (9)$$

The necessary and sufficient conditions on the primary input vector $\vec{T}$ such that the functions $\Omega_k Y$ and $\Omega_{i,j} Y$ are dependent on the line variables L_k and, L_i and L_j, respectively, is that Equations (8) and (9) equal to 1, respectively.

The complete test set for a specific single fault L_k (s-a-a_k), $a_k \in \{0,1\}$, of Equation (8) is defined as [24]:

$$\beta [L_k(\text{s-a-}a_k)] = \{t \mid L_k^* \cdot \Omega_k Y = 1\} \qquad (10)$$

where, $L_k^* = \bar{L}_k$ if $a_k = 1$, and

$L_k^* = L_k$ if $a_k = 0$

For the case of a combinational circuit with a multiple fault involving p internal lines, x1, x2, ..., xp, simultaneously, Das [15] developed much simpler expressions for Boolean difference which require much less computation. The general form for the Boo-

lean difference of Y with respect to any subset of p variables, $L_{x1}, L_{x2}, ..., L_{xp}$, can be now defined as follows:

$$\Omega_{x1,x2,...,xp} Y = \sum_{j=0}^{j=2^{p-1}-1} ({}^{j}m + {}^{2^p-1-j}m)({}^{j}Y \oplus {}^{2^p-1-j}Y) \qquad (11)$$

where ${}^{r}m$ = p-tuple of decimal equivalent r for $\vec{L} = [L_{x1}, L_{x2}, ..., L_{xp}]$, and

${}^{z}Y = Y[\vec{T}$, p-tuple of decimal equivalent z for $\vec{L}]$

The complete set of type I tests to detect 2^p possible simultaneous multiple faults on these p lines is then obtained by making Equation (11) equal to one.

Let us consider an example in [24]:

$$Y(u_1,u_2,u_3,u_4) = \bar{u}_1u_2 + u_2\bar{u}_3u_4 + u_1\bar{u}_2u_3 + u_1\bar{u}_4$$

A double fault involving lines 2 and 3 is considered :

$$L_2 = \bar{u}_1 \qquad L_3 = \bar{u}_3$$

Y can then be expressed in terms of L_2 and L_3 as follows :

$$Y(L_2,L_3) = L_2u_2 + L_3u_2u_4 + \bar{L}_2\bar{L}_3\bar{u}_2 + \bar{L}_2\bar{u}_4$$

Then,

$$\Omega_{2,3}\, Y = \sum_{j=0}^{j=1} ({}^{j}m + {}^{3-j}m) ({}^{j}Y \oplus {}^{3-j}Y)$$

$$= (\bar{L}_2 \bar{L}_3 + L_2 L_3) [\, Y (\bar{L}_2, \bar{L}_3) \oplus Y (L_2, L_3) \,]$$
$$+ (\bar{L}_2 L_3 + L_2 \bar{L}_3) [\, Y (\bar{L}_2, L_3) \oplus Y (L_2, \bar{L}_3) \,]$$
$$= (u_1 u_3 + \bar{u}_1 \bar{u}_3) [\, (\bar{u}_2 + \bar{u}_4) \oplus u_2 \,]$$
$$+ (u_1 \bar{u}_3 + \bar{u}_1 u_3) [\, (u_2 u_4 + \bar{u}_4) \oplus u_2 \,]$$

The complete type-I test set is { 0, 1, 2, 5, 8, 10, 11, 15 }.

For the case of a multiple fault involving p internal variables simultaneously of a sequential circuit being in initial state $\vec{S}(0) = SI$, in an attempt to find the single test vectors, or type I or II tests, the Boolean difference is first applied to the output vector of the iterative combinational circuit, or Equation (3) :

$$\Omega_{x1,x2,\ldots,xp} \vec{O} = \sum_{i=1}^{n} G_i [\, \vec{I}, \vec{S}, \vec{L} \,] \oplus G_i [\, \vec{I}, \vec{S}, \vec{\bar{L}} \,] \qquad (12)$$

where n is the number of primary outputs,

$\vec{L} = [\, L_{x1}, L_{x2}, \ldots, L_{xp} \,]$,

$\vec{\bar{L}} = [\, \bar{L}_{x1}, \bar{L}_{x2}, \ldots, \bar{L}_{xp} \,]$, and

the time t=0 being understood, is dropped from the equation.

Note that the vector Boolean differences in Equation (12) are equal to zero only if every term in the sum is equal to zero.

The residual function of $G_i[\vec{I},\vec{S},\vec{L}]$ can be defined as follows :

$${}^0G_i = {}^0G_i[\vec{I},\vec{S}] = G_i[\vec{I},\vec{S},0,0,\ldots,0],$$

$${}^1G_i = {}^1G_i[\vec{I},\vec{S}] = G_i[\vec{I},\vec{S},0,0,\ldots,1],$$

.

.

.

$${}^rG_i = {}^rG_i[\vec{I},\vec{S}, \text{p-tuple of decimal equivalent r}],$$

.

.

.

$${}^{2^p-1}G_i = {}^{2^p-1}G_i[\vec{I},\vec{S},1,1,\ldots,1].$$

Using the residual functions of $G_i[\vec{I},\vec{S},\vec{L}]$ in the simpler expression by [15], the following simplified form of Equation (12) can then be obtained as :

$$\Omega_{x1,x2,\ldots,xp}\vec{O} = \sum_{j=0}^{j=2^{p-1}-1} ({}^jm + {}^{2^p-1-j}m)\left[\sum_{i=1}^{n} ({}^jG_i \oplus {}^{2^p-1-j}G_i)\right] \qquad (13)$$

where ${}^rG_i = G_i[\vec{I},\vec{S},$ p-tuple of decimal equivalent of r], and

${}^sm$ = p-tuple of decimal equivalent of s

Theorem 1 : Consider the iterative combinational circuit which results from a given synchronous sequential circuit. The complete test set of Type I to detect 2^p possible simultaneous p^{th}-order multiple faults on the p lines, x1, x2,..., xp is :

$$\beta\vec{O}\,[\,L_{x1},L_{x2},...,L_{xp}\,] = \{\,t \mid \Omega_{x1,x2,...,xp}\vec{O} = 1\}$$

$$= \{\,t \mid \sum_{j=0}^{2^{p-1}-1} ({}^{j}m + {}^{2^{p}-1-j}m)\,[\sum_{i=1}^{n} ({}^{j}G_i \oplus {}^{2^{p}-1-j}G_i)\,] = 1\,\} \qquad (14)$$

Corollary 1: A complete test set of Type I to detect the specified multiple fault L_{x1} (s-a-a_{x1}), ..., L_{x2} (s-a-a_{x2}), ..., L_{xp} (s-a-a_{xp}) on the p lines, x1, x2, ..., xp, is :

$$\beta\vec{O}[\,L_{x1}(\text{s-a-}a_{x1}), L_{x2}(\text{s-a-}a_{x2}), ..., L_{xp}(\text{s-a-}a_{xp})]$$

$$= \{\,t \mid L_{x1}(\bar{a}_{x1})\,L_{x2}(\bar{a}_{x2})\,...L_{xp}(\bar{a}_{xp})\;\; \Omega_{x1,x2,...,xp}\vec{O} = 1\,\}$$

$$= \{\,t \mid {}^{r}m\;\; [\sum_{i=1}^{n} {}^{r}G_i \oplus {}^{2^{p}-1-r}G_i\,] = 1\,\} \qquad (15)$$

where, if $a_{xj} = 0$, $L_{xj}(a_{xj}) = L_{xj}$ and $L_{xj}(\bar{a}_{xj}) = \bar{L}_{xj}$,

if $a_{xj} = 1$, $L_{xj}(a_{xj}) = \bar{L}_{xj}$ and $L_{xj}(\bar{a}_{xj}) = L_{xj}$, and

${}^{r}m$ is the p-tuple $L_{x1}(\bar{a}_{x1})\,L_{x2}(\bar{a}_{x2})...\,L_{xp}(\bar{a}_{xp})$ of decimal equivalent r.

To find the complete set of Type I and Type II tests for all possible faults, single through p^{th}-order on the p lines, x1, x2,..., xp, we must compute the vector Boolean difference for all of the n outputs with respect to all 2^p different combinations of the p variables $L_{x1}, L_{x2}, ..., L_{xp}$.

Theorem 2 : Consider the iterative combinational circuit which results from a given synchronous sequential circuit. The complete set of Type I and Type II tests for all possible fault situations from single through p^{th}-order on the p lines, x1, x2, ..., xp, is :

$$\alpha\vec{O}\,[L_{x1},L_{x2},...,L_{xp}] = \{\, t \mid \sum_{i=1}^{n} \Omega_{x1}\vec{O}_i + \Omega_{x2}\vec{O}_i + ... + \Omega_{xp}\vec{O}_i + \Omega_{x1,x2}\vec{O}_i + ... + \Omega_{xp-1,xp}\vec{O}_i \;.\;.\; + \Omega_{x1,x2,..xp}\vec{O}_i = 1 \,\} \qquad (16)$$

Corollary 2 : A complete set of Type I and Type II tests for all fault situations up to and including a maximum of p simultaneous specified faults, L_{x1} (s-a-a_{x1}), L_{x2} (s-a-a_{x2}),..., L_{xp} (s-a-a_{xp}) on the p lines, x1, x2, ..., xp is:

$$\alpha\vec{O}\,[L_{x1}(\text{s-a-}a_{x1}), L_{x2}(\text{s-a-}a_{x2}),..., L_{xp}(\text{s-a-}a_{xp})]$$

$$= \{\, t \mid \sum_{i=1}^{n} L_{x1}(\bar{a}_{x1})\,\Omega_{x1}\vec{O}_i + L_{x2}(\bar{a}_{x2})\,\Omega_{x2}\vec{O}_i + ... + L_{xp}(\bar{a}_{xp})\,\Omega_{xp}\vec{O}_i + L_{x1}(\bar{a}_{x1})L_{x2}(\bar{a}_{x2})\,\Omega_{x1,x2}\vec{O}_i + ... + L_{xp-1}(\bar{a}_{xp-1})L_{xp}(\bar{a}_{xp})\,\Omega_{xp-1,xp}\vec{O}_i + ... + L_{x1}(\bar{a}_{x1})L_{x2}(\bar{a}_{x2})...L_{xp}(\bar{a}_{xp})\,\Omega_{x1,x2,...,xp}\vec{O}_i = 1 \,\} \qquad (17)$$

Since the problem of interest when applying Boolean difference on output or next-state vector is to detect the effect of the specified multiple fault by the output or next-state vector difference, respectively, between the fault-free and faulty circuits, so any fault excitement by either Type I or Type II input vectors is essential and should be considered. Therefore, in this work, wherever Boolean difference finds its application in determining the complete set of input vectors, it is the set of both Type I and Type II vectors.

If Equation (17) is equal to zero (only if every term in the sum is equal to zero), i.e. if no type I and II test could be found, it means that the specified fault has no effect on any of the n primary outputs whenever both fault-free and faulty circuits are in the same present state. Therefore the fault cannot be detected yet by the single input vectors since both circuits are assumed to start in the same initial state. This situation results in Case I.

For case I, the fault may then only be detected when the two circuits are in two different present states at time $t > 0$. This pair of different present states is called detecting state pair which will be discussed in Chapter 4. The problem then is to find a shortest input sequence to bring the two circuits from the same initial state to the first detecting state pair, if one exists. This process is carried out by a detecting tree discussed in Chapter 5. But in order for these to happen, the two circuits must first be able to migrate to two different next states from same present state. This is determined by applying the Boolean difference to the next state vector of the iterative combinational circuit, or Equation (17) with $\vec{O}$ replaced by $\vec{S}^{+}$ for $\vec{S}$ (t+1) of Equation (4) as follows:

$$\alpha\vec{S}^{+}\,[\,L_{x1}(s\text{-}a\text{-}a_{x1}),\,L_{x2}(s\text{-}a\text{-}a_{x2}),\ldots,\,L_{xp}(s\text{-}a\text{-}a_{xp})\,]$$

$$= \{\,t \mid \sum_{j=1}^{b} L_{x1}(\bar{a}_{x1})\;\Omega_{x1}\vec{S}_j^{+} + L_{x2}(\bar{a}_{x2})\;\Omega_{x2}\vec{S}_j^{+} + \ldots + L_{xp}(\bar{a}_{xp})\;\Omega_{xp}\vec{S}_j^{+}$$

$$+ L_{x1}(\bar{a}_{x1})\,L_{x2}(\bar{a}_{x2})\;\Omega_{x1,x2}\vec{S}_j^{+} + \ldots + L_{xp-1}(\bar{a}_{xp-1})\,L_{xp}(\bar{a}_{xp})\;\Omega_{xp-1,xp}\vec{S}_j^{+} + \ldots$$

$$+ L_{x1}(\bar{a}_{x1})\,L_{x2}(\bar{a}_{x2})\ldots L_{xp}(\bar{a}_{xp})\;\Omega_{x1,x2,\ldots,xp}\vec{S}_j^{+} = 1\,\} \qquad (18)$$

If Equation (18) is equal to zero, it means that the specified fault has no effect on any of the b next-state functions; therefore it is undetectable. Note that this is the first early checkpoint for identifying undetectable faults that this new method characterizes.

For each output term in a set of q, where $q \leq n$ and n is the number of primary outputs, whose Boolean difference resulted from Equation (17) is not equal to zero, we then have to substitute the present state vector $\vec{S}$ in that result with the known initial state $\vec{S}(0) = SI$ to see if the specified fault can be detected by a single test vector when the CUT is first started. If these substitutions make all of the q results equal to zero, it is obvious that, because of the initial state setting to SI, the fault effects on all of the q primary outputs are cancelled out. Therefore the fault cannot be detected yet by the single input vectors. This situation results in Case II.

For both case I with Equation (18) not being equal to zero and case II, a test sequence of length greater than one may be found by using transition matrix technique and detecting tree as explained in Chapters 4 and 5. Otherwise, if the Boolean difference of any output term in the set of q from Equation (17) is not equal to zero after having SI

substituted for $\vec{S}$, single test vectors can be found by resolving the input vector to make the result equal to one.

Example 1 : Consider the double fault L_{10} (s-a-1), L_{12} (s-a-0) in Figure 2.1 with the initial state SI = S_1S_2 = 00. From Figure 2.2, the following equations are derived :

$$O_1 = S_1S_2$$
$$O_2 = \bar{I}_2$$
$$S_1^+ = I_1 \bar{S}_1 + \bar{I}_1 S_1$$
$$S_2^+ = L_{12} \bar{S}_2 + L_{10} \bar{S}_1 \bar{S}_2 + \bar{L}_{10} \bar{L}_{12} S_2 + \bar{L}_{12} S_1S_2$$
$$L_{10} = \bar{I}_2$$
$$L_{12} = I_2S_1$$

Since the two output functions O_1 and O_2 do not depend on the two faulty lines L_{10} and L_{12}, their Boolean differences with respect to L_{10} and L_{12} are equal to zero. So, this is case I:

$$\alpha O_1 [L_{10}(\text{s-a-1}), L_{12}(\text{s-a-0})] = 0$$
$$\alpha O_2 [L_{10}(\text{s-a-1}), L_{12}(\text{s-a-0})] = 0$$

The Boolean differences for next-state vector are then checked to see if the fault is undetectable :

$$\alpha S_1^+ [\, L_{10}(\text{s-a-1}), L_{12}(\text{s-a-0}) \,] = 0$$

$$\alpha S_2^+ [\, L_{10}(\text{s-a-1}), L_{12}(\text{s-a-0}) \,] =$$

$$= \bar{L}_{10}\, \Omega_{10}\, F_2 + L_{12}\, \Omega_{12}\, F_2 + \bar{L}_{10}\, L_{12}\, \Omega_{10,12}\, F_2$$

$$= \bar{L}_{10}\, ({}^0F_2 \oplus {}^1F_2) + L_{12}\, ({}^0F_2 \oplus {}^1F_2) + \bar{L}_{10}\, L_{12}\, ({}^1F_2 \oplus {}^2F_2)$$

$$= \bar{L}_{10} [\, \overline{(L_{12}\, \bar{S}_2 + \bar{L}_{12}\, S_1\, S_2)}\, (\bar{S}_1\, \bar{S}_2 \oplus \bar{L}_{12}\, S_2) \,] +$$

$$L_{12} [\, \overline{(L_{10}\, \bar{S}_1\, \bar{S}_2)}\, ((\bar{L}_{10}\, S_2 + S_1\, S_2) \oplus \bar{S}_2) \,] +$$

$$\bar{L}_{10}\, L_{12} [\, \bar{S}_2 \oplus (\bar{S}_1\, \bar{S}_2 + S_1\, S_2) \,]$$

$$= I_2$$

Since the Boolean difference for S_2^+ is not equal to zero, there may be a test sequence of length greater than one for this fault. This will be studied in the next two chapters.

Example 2 : Consider another double fault L_4 (s-a-1), L_{13} (s-a-0), with initial state SI = $S_1S_2 = 00$. The functions for outputs, next states and faulty lines are as follows :

$$O_1 = L_4\, S_2$$

$$O_2 = \bar{I}_2$$

$$S_1^+ = I_1 \oplus S_1$$

$$S_2^+ = S_2 \oplus (L_{13} + I_2\, S_1)$$

$$L_4 = S_1$$

$$L_{13} = \bar{L}_2\, \bar{S}_1$$

The Boolean difference is first applied to the output vector :

$$\alpha O_2 [\, L_4(s\text{-}a\text{-}1), L_{13}(s\text{-}a\text{-}0) \,] = 0$$

$$\alpha O_1 [\, L_4(s\text{-}a\text{-}1), L_{13}(s\text{-}a\text{-}0) \,] =$$

$$= \bar{L}_4 \Omega_4 G_1 + L_{13} \Omega_{13} G_1 + \bar{L}_4 L_{13} \Omega_{4,13} G_1$$

$$= \bar{L}_4 ({}^0G_1 \oplus {}^1G_1) + L_{13} ({}^0G_1 \oplus {}^1G_1) + \bar{L}_4 L_{13} ({}^1G_1 \oplus {}^2G_1)$$

$$= \bar{L}_4 (S_2 \oplus 0) + L_{13} (0 \oplus 0) + \bar{L}_4 L_{13} (S_2 \oplus 0)$$

$$= \bar{L}_4 S_2$$

$$= \bar{S}_1 S_2$$

The Boolean difference for output O_1 is not equal to zero, and thus we next substitute its present-state vector with the initial state $SI = S_1S_2 = 00$. This substitution certainly cancels out the fault effect on O_1 by causing the Boolean difference to be equal to zero, resulting in case II. We will continue in Chapters 4 and 5 to look for the shortest test sequence for this fault.

Chapter IV
TRANSITION AND DETECTION MATRICES

4.1 Transition Matrix

For the case in which the application of the Boolean difference method described in the previous chapter indicates that there is no single test vector for the specified multiple fault, another technique should be used to search for the shortest test vector of length greater than one. In order to do that, we propose a method that needs to know about the behavior of the fault-free and faulty circuits in different present states and under different input vectors. This information is very well provided in the state tables. Instead of using the state tables, the present method makes use of the transition matrix [13], which was developed to study the measurement and control problems of synchronous sequential machines, and used here with a small change to include both the fault-free and faulty circuits. The reason for this representation is that this matrix visually eases the process of forming the detection matrix defined below, and if this process is carried out using a computer, the matrix form is very systematic and straightforward, and hence readily implementable.

For a given sequential circuit under test with a multiple fault π, the transition matrix [T] is formed with column and row corresponding to present states and inputs, respectively, and each entry is represented in the form (fault-free next state, fault-free output O; fault-π next state, fault-π output O_π), where states are denoted by letters and inputs and outputs by their decimal equivalents. If there are b flip-flops and m primary inputs, [T] is of size $2^b \times 2^m$.

The transition matrices for Examples 1 and 2 in previous chapters are shown in Figures 4.1(a) and 4.2(a), respectively.

From Figure 4.2(a), it is obvious that this is case I since there is no output difference between the fault-free and faulty circuits for every entry. Case II can be justified in Figure 4.2(a), since there is no output difference for every entry only of row A which is the initial state.

4.2 Detection Matrix

Assume the fault-free and fault-π sequential circuits start in the same initial state SI.

Definition 6 : Suppose that at time $t \geq 0$ the fault-free and fault-π circuits are in state $\vec{S}_{ff}(t) = S_x$ and $\vec{S}_\pi(t) = S_y$, respectively. The state S_y is said to be a detecting state for fault π, if there is at least one input vector which when applied can produce a difference in output vector between the two circuits at time (t+1). The pair (S_x, S_y) is then called a detecting pair of states, or shortly detecting pair, where S_x and S_y may be the same.

The set of detecting pairs for fault π is represented in a matrix [D] called state detection matrix, or just detection matrix, in which the column and row correspond to the present states of fault-free and fault-π circuits, respectively. The entry [S_i , S_j], common to row S_i and column S_j, is in the form (I ; O, O_π) if and only if, there exists an input I which when applied to both fault-free and fault-π circuits which are in present states S_i and S_j, respectively, will produce outputs O and O_π, respectively. Of course, O and O_π must be different. Note that there might be no entry for some [S_i , S_j] and some may have more than one such testing inputs, but only one is needed and entered in [D]. It is obvious that if [D] is empty, the fault is certainly undetectable. Therefore, this is another checkpoint for identifying undetectable faults that this new method characterizes before proceeding any further. But it is also very important to note that, even if [D] is not empty, a test sequence of length greater than one is still not guaranteed to be found for the fault π until we build the detecting tree which is explained in the next chapter.

The procedure to form the detection matrix [D] from the transition matrix [T] is as follows:

Let O_i and $O_{\pi i}$ be the fault-free and fault-π outputs respectively at row i in [T]. For each column j of the 2^b columns, go through each row i of the 2^m rows. If O_i differs from:

- $O_{\pi i}$, a detecting pair (S_i , S_i) is obtained, then entry [S_i , S_i] of [D] is entered with (j ; O_i , $O_{\pi i}$).
- $O_{\pi k}$ at each row k of the other 2^b - 1 rows, a detecting pair (S_i , S_k) is obtained, then entry [S_i , S_k] is entered with (j ; O_i , $O_{\pi k}$).

The detection matrices for Examples 1 and 2 are shown in Figures 4.1(b) and 4.2(b), respectively.

			Primary Inputs			
	S_1 S_2		0	1	2	3
	0 0	A	B,1 ; B,1	A,0 ; B,0	D,1 ; D,1	C,0 ; D,0
Present	0 1	B	A,1 ; A,1	B,0 ; A,0	C,1 ; C,1	D,0 ; C,0
States	1 0	C	C,1 ; C,1	D,0 ; C,0	A,1 ; A,1	B,0 ; A,0
	1 1	D	D,3 ; D,3	C,2 ; D,2	B,3 ; B,3	A,2 ; B,2

(a)

		Faulty States			
		A	B	C	D
Fault-	A				0;1,3
Free	B				1;0,2
States	C				2;1,3
	D	0;3,1	2;3,1	3;2,0	

(b)

Figure 4.1 (a) [T] and (b) [D] for L_{10} (s-a-1), L_{12} (s-a-0)

	0	1	2	3
A	B,1 ; A,1	A,0 ; A,0	D,1 ; C,1	C,0 ; C,0
B	A,1 ; B,3	B,0 ; B,2	C,1 ; D,3	D,0 ; D,2
C	C,1 ; C,1	D,0 ; D,0	A,1 ; A,1	B,0 ; B,0
D	D,3 ; D,3	C,2 ; C,2	B,3 ; B,3	A,2 ; A,2

(a)

	A	B	C	D
A		1;0,2		0;1,3
B		0;1,3		3;0,2
C		1;0,2		2;1,3
D	0;3,1		3;2,0	

(b)

Figure 4.2 (a) [T] and (b) [D] for L_4(s-a-1), L_{13}(s-a-0)

Chapter V
DETECTING TREE

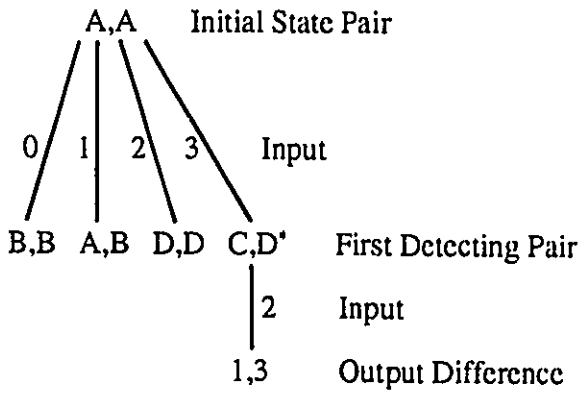
The detecting tree developed here borrows the concept of the distinguishing tree of sequential machine [14], but brings both the fault-free and faulty circuits, side by side from time to time under the same sequence of input vectors to show their behaviors until there is an evidence of a difference in output vector between the two circuits, if one exists. Goldstein [20] used a similar successor tree for keeping track of the states reached and the outputs generated by both circuits.

In the detecting tree, each node is a pair of fault-free and faulty states being reached by the same input sequence applied to both circuits. The tree is started at level 0 with only one node which is (SI , SI) since both circuits are assumed to start in the same known initial state SI. The tree is then branched out from each node at level i by exhaustively generating more nodes at level i+1 under 2^m input vectors, according to the behaviors of both circuits shown in the transition matrix [T]. Each new node in the tree is checked to see if it is terminal , i.e. cannot generate any node, according to the termination rule [13], that is, it has already appeared in the tree. The terminal is marked with a double underline. For each new non-terminal node, the detection matrix [D] is checked to see if this node is a detecting pair. If it is, and the first in the tree, mark it with a '*',

then a test is found by the shortest sequence of length i+1, consisting of the input vector of length i leading to this node followed by the single vector shown at the corresponding entry for that detecting pair in [D]. The final, and of course different, fault-free and faulty outputs are also shown at that entry. This process is repeated at level i+1 when there are no more generating nodes to work with at level i.

This whole process is terminated when either the first test has been found, or the tree cannot branch out anymore because all nodes at the current level are terminals, or the tree has reached its predefined maximum level. For the last two cases, the fault is said to be undetectable.

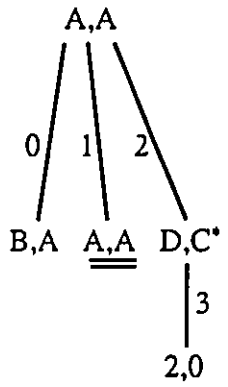
Again, being back to Examples 1 and 2, their detecting trees are shown in Figures 5.1 and 5.2, respectively.



Solution : test sequence of length 2 : 3, 2

with final fault-free and faulty outputs : 1,3, respectively.

Figure 5.1 Detecting tree for L_{10} (s-a-1), L_{12} (s-a-0)



Solution : test sequence of length 2 : 2,3

with final fault-free and faulty outputs : 2,0, respectively.

Figure 5.2 Detecting tree for L_4 (s-a-1), L_{13} (s-a-0)

Chapter VI

APPLICATION AND EXPERIMENTAL RESULTS

One of the most important aspects in the development of a test generation method is the ability to make its implementation automatic, by effective and efficient programming. Since Boolean difference for single fault can be found by rotating the Karnaugh map [12], (as shown in Appendix A) - method for finding Boolean difference for multiple faults by using Karnaugh map, or any other similar representation, is shown. When implemented on a computer, this saves us from all the works with formulas and computations described in the preceding chapters. Based on the systematic representations of map and matrix, and on the algorithmic implementations of Boolean difference, matrix computation and tree configuration, an effective computer program, written in C language, is developed for the new method presented in this dissertation to verify its capability as a basic tool for an automatic test generator. This program, which is included in Appendix B, can generate test sequences for detecting single or multiple faults in both combinational and synchronous sequential circuits.

In what follows, experimental applications to both Moore and Mealy sequential circuits are presented.

6.1 Example of Testing a Moore Sequential Circuit

Example 3 : Testing type '164 IC - 8-bit parallel-out shift registers

The functional block diagram is shown in Figure 6.1 [38]. Since this is a Moore circuit, i.e. all the outputs are functions of the present states only, output vector can be checked in a current time frame without setting input vector. In other words, if there are i time frames needed to detect a fault, the test sequence of only (i-1) input vectors is required.

From the iterative combinational circuit in Figure 6.2, with a triple fault on lines 11, 18, and 20, the following equations can be obtained :

- Functions of faulty lines :

$$L_{11} = \bar{Q}_B \,,\; L_{18} = Q_C \,,\; L_{20} = Q_D$$

- Functions of outputs :

$$F_A = Q_A \,,\; F_B = Q_B \,,\; F_C = Q_C \,,\; F_D = L_{20}$$
$$F_E = Q_E \,,\; F_F = Q_F \,,\; F_G = Q_G \,,\; F_H = Q_H$$

- Functions of next states :

$$Q_A^+ = AB,\; Q_B^+ = Q_A \,,\; Q_C^+ = \bar{L}_{11}\, Q_C + Q_B \,,\; Q_D^+ = L_{18} + L_{20}\, Q_C \,,$$
$$Q_E^+ = L_{20} + Q_E\, Q_D \,,\; Q_F^+ = Q_E \,,\; Q_G^+ = Q_F \,,\; Q_H^+ = Q_G$$

- Binary representations of :

  Input Vector = AB

  Output Vector = $F_A F_B F_C F_D F_E F_F F_G F_H$

  State Vector = $Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H$

  Faulty-Line Vector = $L_{11} L_{18} L_{20}$

- Assume control lines CLEAR and CLOCK are fault-free
- The circuit is reset with CLEAR=0 and then back to 1, to have it starting in initial state 0.

Results from the program running with IBM CPU 3090 as shown in Appendix C:

$L_{11} L_{18} L_{20}$ s-a-	Test Length : Testing Sequence	Fault-Free, Faulty Outputs	CPU Time Used
0 0 1	0 : RESET	0,16	224 msec
0 0 0	4 : 3 0 0 0	16,32	816 msec

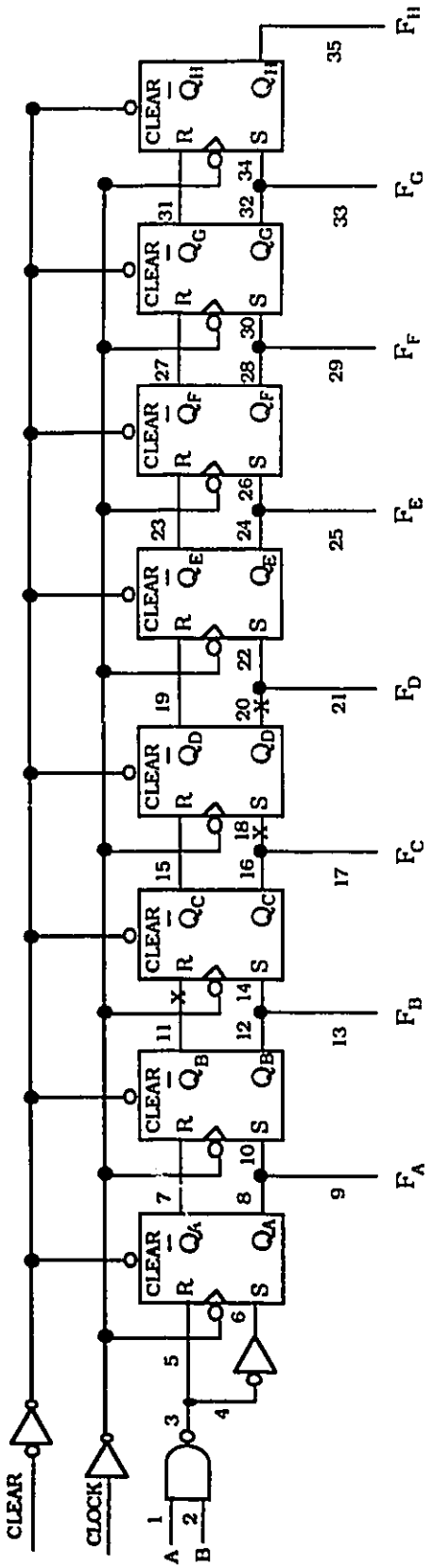


Figure 6.1 Type '164 IC- 8-bit parallel-out serial shift register

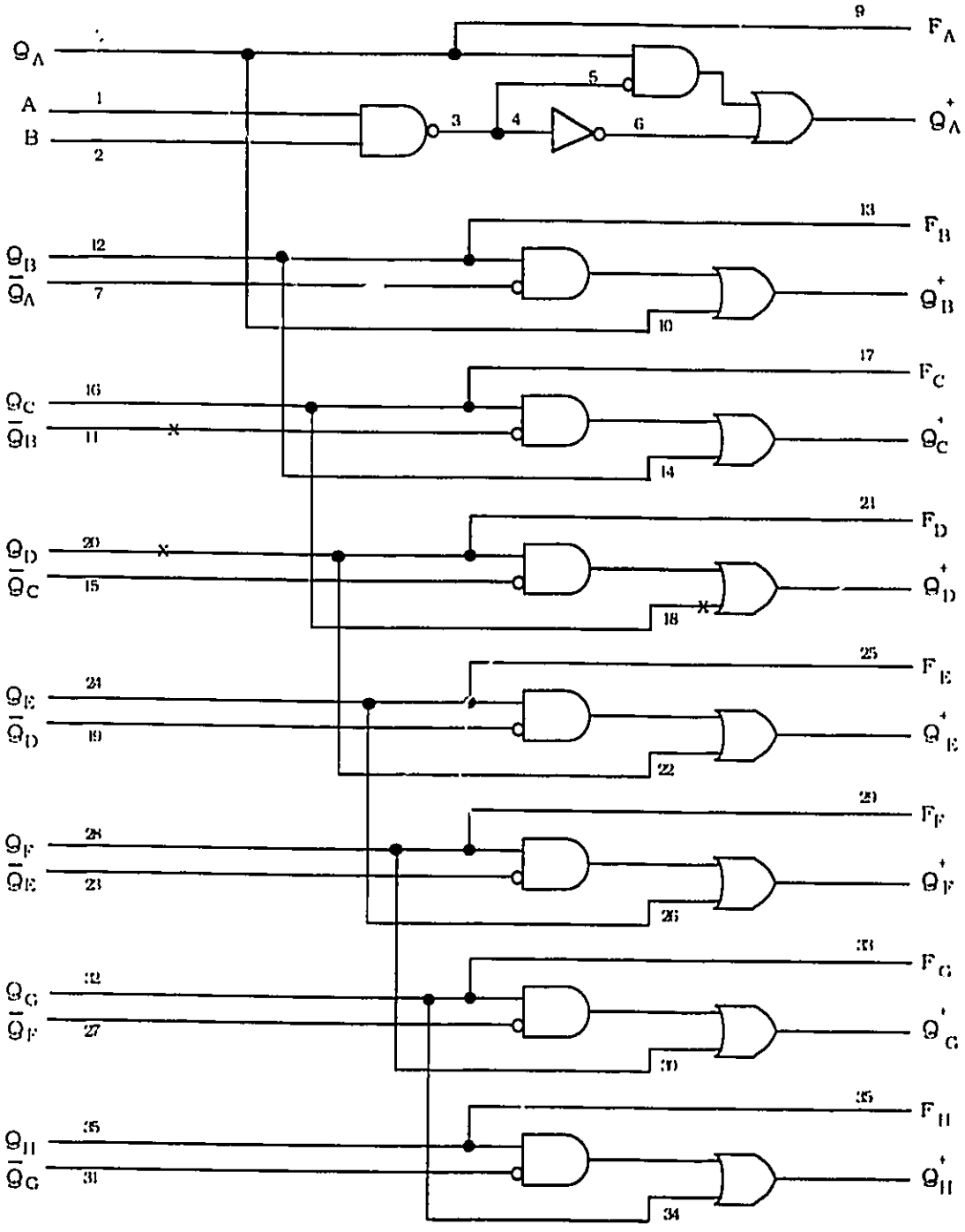


Figure 6.2 Iterative Combinational Circuit for Figure 6.1

6.2 Example of Testing a Mealy Sequential Circuit

Example 4 : Testing type '97 IC - Synchronous 6-bit binary rate multiplier.

The functional block diagram is shown in Figure 6.3 [38]. Since this is a Mealy circuit, i.e. at least one of the primary outputs is a function of both input and present-state vectors, there are i time frames needed to detect a fault by a test sequence of length i.

In Figure 6.4, consider a triple fault on lines 6, 16 and 19 :

- Functions of faulty lines :

$$L_6 = C\ ,\ L_{16} = Q_B\ ,\ L_{19} = \overline{Q}_C$$

- Functions of outputs :

$$W = \overline{Q}_A + \overline{Q}_B + \overline{Q}_C + \overline{Q}_D + \overline{Q}_E + \overline{Q}_F$$

$$Y = \overline{K}(F\overline{Q}_A + EQ_A\ \overline{Q}_B + DQ_A\ Q_B\ L_{19} + L_6\ Q_A\ Q_B\ Q_C\ \overline{Q}_D +$$
$$BQ_A\ Q_B\ Q_C\ Q_D\ \overline{Q}_E + AQ_A\ Q_B\ Q_C\ Q_D\ Q_E\ \overline{Q}_F)$$

$$Z = K + [\ \overline{(F\overline{Q}_A)}\ .\ \overline{(EQ_A\ \overline{Q}_B)}\ .\ \overline{(DQ_A\ Q_B\ L_{19})}\ .\ \overline{(L_6\ Q_A\ Q_B\ Q_C\ \overline{Q}_D)}\ .$$
$$\overline{(BQ_A\ Q_B\ Q_C\ Q_D\ \overline{Q}_E)}\ .\ \overline{(AQ_A\ Q_B\ Q_C\ Q_D\ Q_E\ \overline{Q}_F)}\ ]$$

- Functions of next states :

$$Q_A^+ = \overline{Q}_A\ ,\ Q_B^+ = Q_A \oplus Q_B\ ,\ Q_C^+ = (L_{16}\ Q_A) \oplus Q_C$$
$$Q_D^+ = Q_A\ Q_B\ Q_C \oplus Q_D\ ,\ Q_E^+ = Q_A\ Q_B\ Q_C\ Q_D \oplus Q_E$$
$$Q_F^+ = Q_A\ Q_B\ Q_C\ Q_D\ Q_E \oplus Q_F$$

- Binary representations of :

  Input Vector = K F E D C B A

  Output Vector = W Y Z

  State Vector = Q_A Q_B Q_C Q_D Q_E Q_F

  Faulty-Line Vector = L_6 L_{16} L_{19}

- Assume control lines CLEAR, CLOCK, STROBE, UNITY/CASCADE, and ENABLE INPUT are fault-free.
- Set STROBE = 1, UNITY/CASCADE = 1, ENABLE INPUT = 0
- Reset the circuit with CLEAR = 1 and back to 0, to have it starting in initial state 0.

Results from the program as shown in Appendix C:

L_6 L_{16} L_{19} s-a-	Test Length : Testing Sequence	Fault-Free, Faulty Outputs	CPU Time Used
0 0 1	8 : 0 0 0 0 0 0 0 4	6,5	2836 msec
1 0 0	4 : 0 0 0 8	5,5	2830 msec

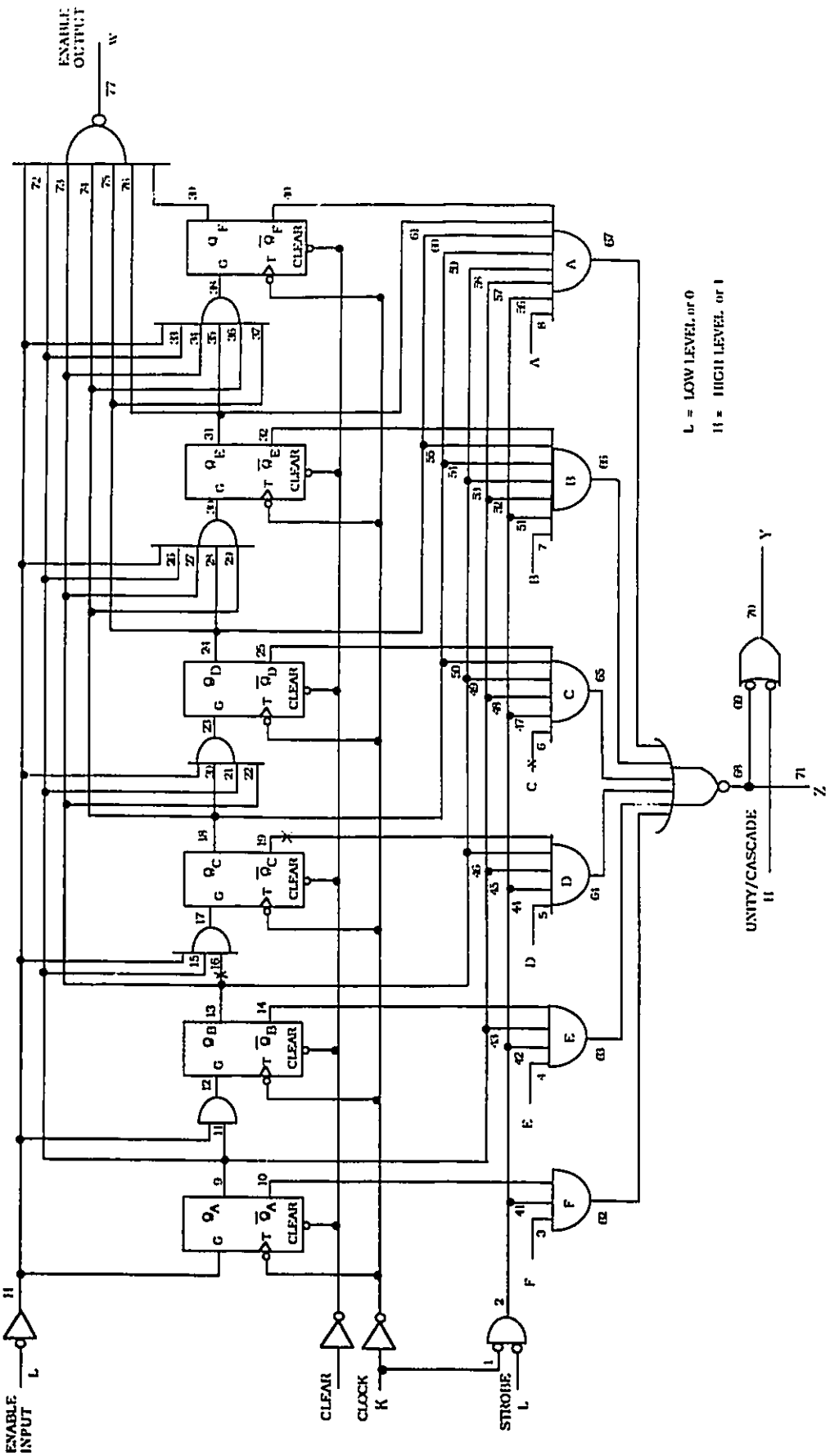


Figure 6.3 Type '97 IC - Synchronous 6-bit binary rate multipliers

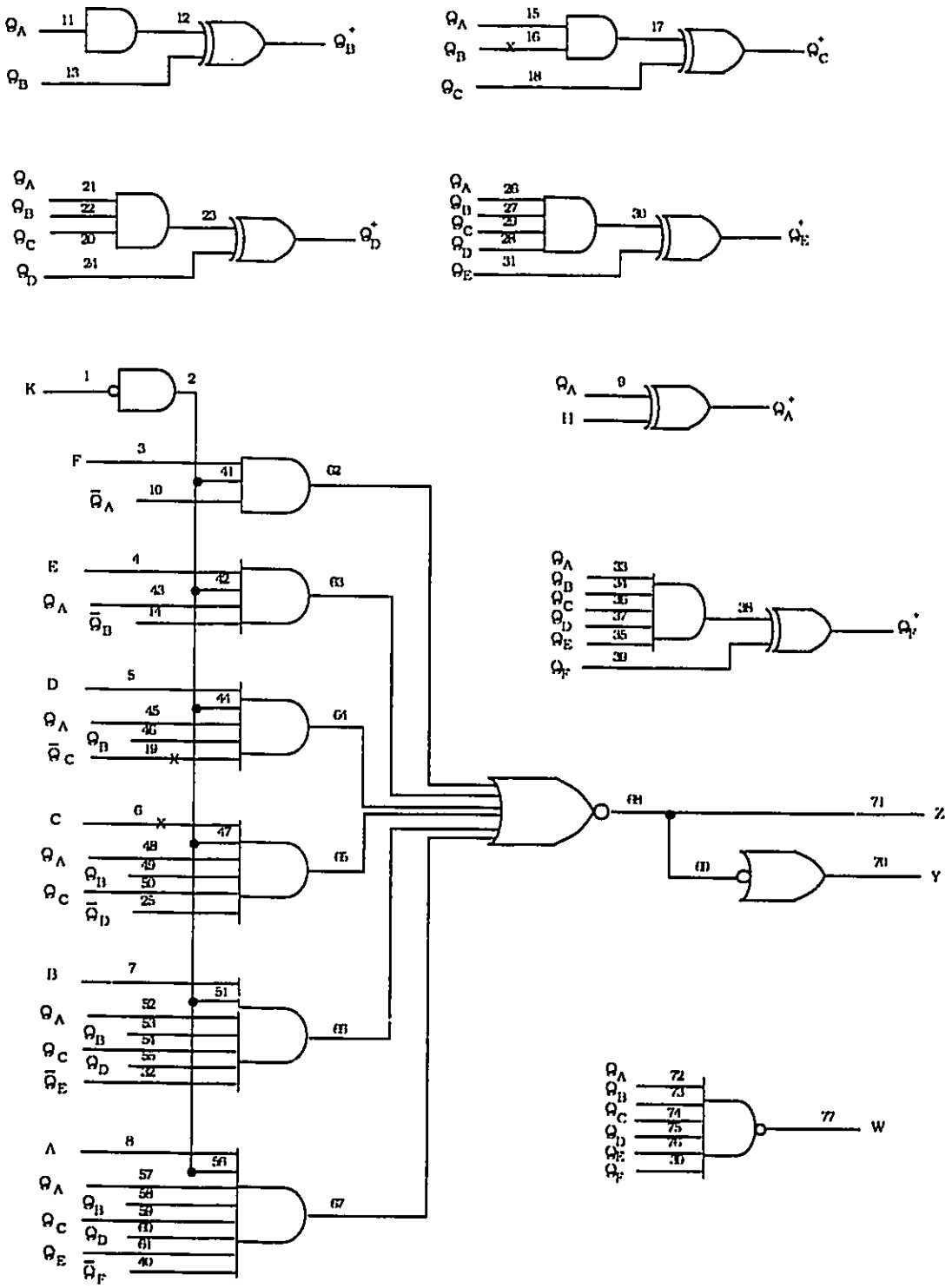


Figure 6.4 Iterative Combinational Circuit for Figure 6.3

Chapter VII
DISCUSSION AND CONCLUSION

In this dissertation, the vector Boolean difference and transition matrix techniques have been utilized in test sequence generation for detecting faults in synchronous sequential circuit. The strong and weak points of the proposed method can be summarized as follows.

- It is complete : It guarantees the generation of a test sequence, if one exists, for every multiple stuck-at fault in a synchronous sequential circuit having a known initial state.

- It is also minimal by guaranteeing shortest test sequence generated. Goldstein's method [20, 21], on which the new method is partially based , cannot guarantee the generation of minimal test sequence. Only the Nine-Value algorithm [31] has claimed this outcome.

- It provides very early recognition of undetectable or redundant faults at two stages : if the Boolean difference indicates that the fault has no effect on both output and next-state vectors; or if later there is no detecting pair found, i.e. the detection matrix is empty. Redundant fault identification is very difficult [1], sometimes, there is no way to do it unless all possible test-input vectors are applied.

- It is very systematic and completely algorithmic, and hence very readily implementable on a computer by the elegant representations of map and matrix. So it has been shown that, even if at first, this new method theoretically appears to involve complex manipulation of algebraic equations, it can be carried out effectively through computer since Boolean difference for multiple fault can be found by using Karnaugh map, or any such similar representation, as shown in Appendix A.

- It requires least involvement of circuit topology : it only needs the equations of the faulty lines besides those of outputs and next-states. In other words, this can save a lot in complexity and memory usage compared to other methods which require line justification and tracking on trial and error basis.

However, since this new method needs a lot of storage for the map, matrices, and tree, the memory allocation saved compared to other methods is trade off for this disadvantage. This is what limits our experimental application to MSI circuits only as shown in Chapter 6. However, if provided with a computer with larger storage besides a better programming technique to minimize the memory storage required, we can make some increase in any of the numbers of inputs, outputs, flip-flops or faulty lines. Overall, it has been shown that the new method proposed herein can be considered as a new tool for an automatic test generator, even though it may not be practical for testing VLSI circuits at this moment due to the memory storage limitation.

BIBLIOGRAPHY

1. M. Abramovici and M. A. Breuer, "On Redundancy and Fault Detection in Sequential Circuits," *IEEE Trans. on Computers*, Vol. C-28, No.11, pp.864-865, Nov. 1979.

2. M. Abramovici, M. A. Breuer and A. D. Friedman, "Digital Systems Testing and Testable Design," Computer Science Press, An Imprint of W.H. Freeman and Company, NY, 1990.

3. V. D. Agrawal, K. T. Cheng and P. Agrawal, "A Directed Search Method for Test Generation Using a Concurrent Fault Simulator," *IEEE Trans. on Computer-Aided Design*, pp.131-138, Feb.1989.

4. V. D. Agrawal and S. C. Seth, "Test Generation for VLSI Chips - Tutorial," IEEE Computer Society Press, Washington D.C., 1988.

5. M. J. Bending, "HITEST - A Knowledge-Based Test Generation System," *IEEE Design and Test of Computers*, Vol. 1, pp. 83-93, May 1984.

6. D. Brahme and J.A. Abraham, "Functional Testing of Microprocessors," *IEEE Trans. on Computers*, Vol. C-33, No. 6, pp.475-485, June 1984.

7. M. A. Breuer and A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, Rockville, MD, 1976.

8. H. T. Cheng and V. D. Agrawal, "Unified Methods for VLSI Simulation and Test Generation," Kluwer Academic Publishers, Norwell, MA, 1989.

9. K. T. Cheng and J. Y. Jou, "Functional Test Generation for Finite State Machines," *Proc. of International Test Conf.*, pp. 162-168, 1990.

10. W. T. Cheng, "The BACK Algorithm for Sequential Test Generation," *Proc. Int. Conf. Computer Design*, (ICCD-88), Rye Brook, NY, pp. 66-69, October 1988.

11. W. T. Cheng and T. J. Chakraborty, "Gentest: an Automatic Test-Generation System for Sequential Circuits," *IEEE Computer*, Vol. 22, pp. 43-49, April 1989.

12. S. R. Das, "Boolean Difference Methods," Notes for Course ELG 5195, University of Ottawa, 1985.

13. S. R. Das, W. J. Hsu, Z. Chen, and W. B. Jone, "Further Studies on the Matrix Approach to the Measurement and Control Problems of Synchronous Sequential

Machines - Performance Evaluation by Computer Simulation and Application of Specific Heuristics," *Comput. and Elect. Engng*, Vol. 12, No. 3/4, pp. 161-173, 1986.

14. S. R. Das, C. L. Sheng, and Z. Chen, "Transition Matrices in the Measurement and Control of Synchronous Sequential Machines," *Information Sciences 18*, pp.47-65, 1979.

15. S. R. Das, P. K. Srimani, and C. R. Dutta, "On Multiple Fault Analysis in Combinational Circuits by Means of Boolean Difference," *Proc. of the IEEE*, Vol. 64, No. 9, pp. 1447-1449, Sept. 1976.

16. H. Fujiwara, "Logic Testing and Design for Testability," The MIT Press Series in Computer Systems, Cambridge, MA, 1985.

17. H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. on Computers*, Vol. C-32, pp. 1137-1144, Dec. 1983.

18. A. Ghosh, S. Devdas and A. R. Newton, "Test Generation and Verification for Highly Sequential Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 10, No. 5, pp. 652-667, May 1991.

19. P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combination Logic Circuits," *IEEE Trans. on Computers*, Vol. C-30, pp.215-222, March 1991.

20. L. H. Goldstein, "Analysis of Multiple Faults in Synchronous Sequential Circuits by Boolean Difference Techniques," SANDIA-77-0709, Sandia Laboratories, Albuquerque, New Mexico, 1977.

21. L. H. Goldstein, "Multiple Fault Analysis in Synchronous Sequential Circuits by Means of Vector Boolean Difference," SANDIA-77-0702J, Sandia Laboratories, Albuquerque, New Mexico, 1977.

22. F. J. Hill and B. Huey, "A Design Language Based Approach to Test Sequence Generation," *Computer*, Vol. 10, pp.28-33, June 1977.

23. T. P. Kelsey and K. W. Saluja, "Fast Test Generation for Sequential Circuits," *Proc. of IEEE International Conf. on Computer-Aided Design*, pp. 354-357, 1989.

24. C. T. Ku and G. M. Masson, "The Boolean Difference and Multiple Fault Analysis," *IEEE Trans. on Computers*, Vol. C-21, No. 1, pp. 62-71, Jan. 1975.

25. H. Kubo, "A Procedure for Generating Test Sequences to Detect Sequential Circuit Failures," *NEC J. Res. and Dev. (12)*, pp.69-78, October 1968.

26. T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 1, pp. 4-15, Jan. 1992.

27. H-K. T. Ma, S. Devdas, A. R. Newton and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 10, pp. 1081-1093, Oct. 1988.

28. R. Malett, "An Efficient Test Generation System for Sequential Circuits," *Proc. of 23rd Design Automation Conf.*, pp. 250-256, 1986.

29. S. Mallela and S. Wu, "Sequential Circuit Testing Generation System," *Proc. of International Test Conf.*, pp. 57-61, Nov. 1985.

30. R. A. Marlett, "EBT: A Comprehensive Test Generation Technique for Highly Sequential Circuits", *Proc. Des. Auto. Conf.*, Las Vegas, Nevada, pp. 335-339, June 1978.

31. P. Muth, "A Nine-Valued Circuit Model for Test Generation," *IEEE Trans. on Computers*, Vol. C-25, No. 6, pp. 630-636, June 1976.

32. I. Pomeranz and S. M. Reddy, "On Achieving a Complete Fault Coverage for Sequential Machines using the Transition Fault Model," *ACM/IEEE Design Automation Conf.*, 1991.

33. G. R. Putzolu and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," *IEEE Trans. on Computers*, Vol. C-20, pp. 639-647, June 1971.

34. J. P. Roth, W. G. Beuricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests and to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. on Electronic Computers*, Vol. EC-16, pp.567-580, October 1967.

35. F. F. Sellers, M. Y. Hsiao, and L.W. Bearnson, "Analyzing Errors with Boolean Difference," *IEEE Trans. on Computers*, Vol. C-17, pp. 676-683, July 1968.

36. T. J. Snethen, "Simulation Oriented Fault Test Generator," *14th Design Automation Conf.*, pp. 88-93, 1977.

37. Y. Takamatsu and K. Kinoshita, "CONT: A Concurrent Test Generation Algorithm," Fault-Tolerant Computing Symp. (FTCS-17) Digest of Papers, Pittsburgh, PA, pp. 22-27, July 1987.

38. "The TTL Data Book for Design Engineers", Texas Instruments Inc., 2nd Edition, 1981.

39. T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *Proceedings of the IEEE*, pp. 98-112, 1983.

PAPERS BY THE AUTHOR ON WHICH PORTIONS OF THE PRESENT DISSERTATION IS BASED

1. S. R. Das, T. V. Nguyen, and A. R. Nayak, "On Multiple Fault Analysis in Synchronous Sequential Circuits by Boolean Difference Techniques," *Proc. of the 32nd Midwest Symposium on Circuits and Systems*, Champaign, IL, 14-16 Aug. 1989, Vol. 1, pp.311-316.

2. S. R. Das, T. V. Nguyen, and A. R. Nayak, "Analyzing Multiple Faults in Synchronous Sequential Circuits by Boolean Difference Techniques," *Cybernetics and Systems: An International Journal*, Vol. 21, pp.461-474, 1990.

3. S. R. Das, W. B. Jone, T. V. Nguyen, and A. R. Nayak, "Detection of Multiple Stuck Faults in Synchronous Sequential Circuits Using Boolean Difference and Transition Matrix Techniques," Paper has been accepted for publication on *IEEE Trans. on Systems, Man, and Cybernetics*.

Appendix A

FINDING BOOLEAN DIFFERENCE BY USING KARNAUGH MAP

In this Appendix, method for finding Boolean difference by using the Karnaugh map, or any such similar representation, is shown.

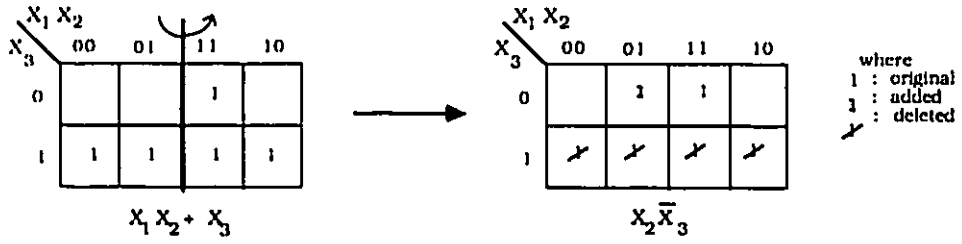
Let us take an example in [12], for a function

$$F(x) = x_1 x_2 + x_3$$

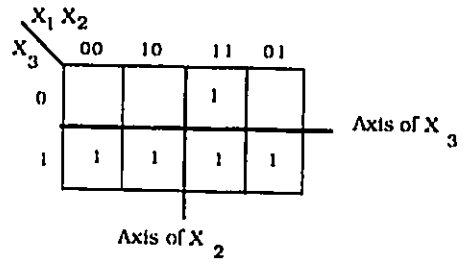
The Boolean difference of F with respect to x_1 is

$$\Omega_{x1} F = \frac{dF(x)}{dx_1} = F(x_1=0) \oplus F(x_1=1) = x_2\bar{x}_3$$

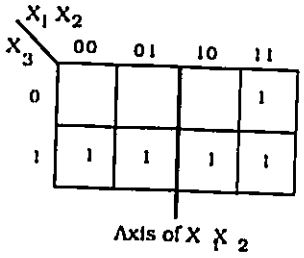
This is equivalent to rotating the Karnaugh map of F(x) about the axis of x_1, while at the same time doing the Exclusive-OR operation on every pair of cells, i.e. a cell in which x_1=0 is exclusive-ored with that in which x_1=1 while keeping x_2 and x_3 the same, or columns 00 ⊕ 10, and 01 ⊕ 11:



Similarly, if we want to find the Boolean difference of F with respect to x_2 and x_3, with the re-arrangement of the columns resulting in a map with similar representation, the axes of x_2 and x_3 are shown in the following map :



For the case of multiple Boolean difference, for example, if we are interested in x_1 and x_2, the axis of x_1,x_2 is shown as:



Let us consider another example in [24]:

$$F(x_1,x_2,x_3,x_4) = \bar{x}_1x_2 + x_2\bar{x}_3x_4 + x_1\bar{x}_2x_3 + x_1\bar{x}_4$$

A double fault in which L_2 (s-a-1) and L_3 (s-a-1) is considered :

$$L_2 = \bar{x}_1 \qquad L_3 = \bar{x}_3$$

F can then be expressed in terms of L_2 and L_3 as follows :

$$F(L_2,L_3) = L_2x_2 + L_3x_2x_4 + \overline{L}_2\overline{L}_3\overline{x}_2 + \overline{L}_2\overline{x}_4$$

Since the above equation now does not depend on x_1 and x_3 any more, the Karnaugh map for $F(L_2, L_3)$ can be reduced by leaving x_1 and x_3 out for simplicity:

L_2L_3 \ x_2x_4	00	01	11	10
00	1	1		1
01	1		1	1
11			1	1
10			1	1

The complete test set (Type I) for $F(L_2 , L_3)$ can be found by rotating about L_2 and L_3, i.e. rows $00 \oplus 11$, and $01 \oplus 10$, to obtain $\Omega_{2,3}\,F$. The result is:

L_2L_3 \ x_2x_4	00	01	11	10
00	1	1	1	
01	1			
11	1	1	1	
10	1			

(a)

$\Omega_{2,3}\,F$

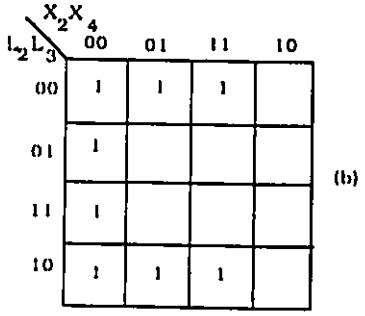
which is {0, 1, 2, 5, 8, 10, 11, 15} after resolving L_2 and L_3 . This is the same as was obtained in [24].

The complete test set (Type I) for double fault L_2 (s-a-1) and L_3 (s-a-1) is just the row 00 of (a), where $L_2L_3 = 00$, or {10, 11, 15}, since we want to drive $L_2L_3 = 00$. This is just $\bar{L}_2\,\bar{L}_3\,\Omega_{2,3}F$.

The complete test set (Type I and II) for $F(L_2,L_3)$ is

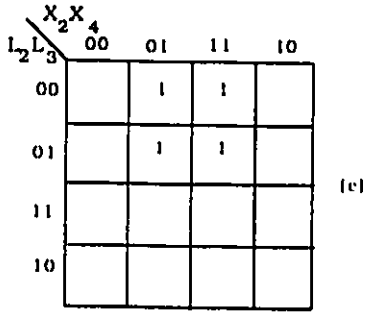
$$\{t \mid \Omega_{2,3}F + \Omega_2F + \Omega_3F = 1\}$$

$\Omega_2 F$ and $\Omega_3 F$ can be found by rotating about L_2 and L_3, respectively :



(b)

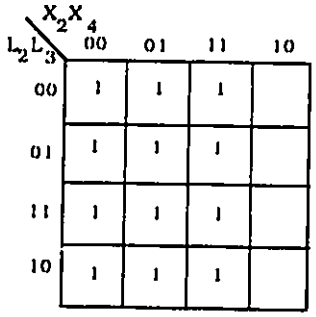
Ω_2F



(c)

Ω_3F

The combination of (a), (b), and (c) is :



which is {0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15}

The complete test set (Type I and II) for double fault L_2 (s-a-1), L_3 (s-a-1) is

$$\{t \mid \overline{L}_2 \cdot \Omega_2 F + \overline{L}_3 \cdot \Omega_3 F + \overline{L}_2 \overline{L}_3 \cdot \Omega_{2,3} F = 1\}$$

This is the combination of row 00 of (a) where $L_2L_3 = 00$, rows 00, 01 of (b) where $L_2 = 0$, and rows 00, 10 of (c) where $L_3 = 0$:

L_2L_3 \ X_2X_4	00	01	11	10
00	1	1	1	
01	1			
11				
10				

which is {8, 10, 11, 15}.

Appendix B

C-PROGRAM LISTING

```
/* Test Generation for Synchronous Sequential Circuits Using */
/* Boolean Difference and Transition Matrix */

#include <stdio.h>
#include <time.h>
#include <string.h>

char seq,mealy,found,bdflag,trflag,sflag;
unsigned short i,j,k,q,w,y,z,m2,n2,x2,b2,nc,pc,detpair,stuckdec;
unsigned m,n,b,x,p,sum,Si;
char pow[8] = {'\1','\2','\4','\10','\20','\40','\100','\200'};
char stuck[4];   /* stuck values vector */
char O[262144];  /* outputs map */
char S[262144];  /* next states map */
char L[16384];   /* stuck lines map */
struct { char binary [4];
         char status;
         char value;
       } A[16];  /* reference matrix */
struct {
         char So;
         char Oo;
         char Sf;
         char Of;
       } T[256][128];  /* transition matrix */
struct {
         char status;
         char I;
         char Oo;
         char Of;
       } D[256][256];  /* detection matrix */
struct {
         char level;
         char status;
         char So;
         char Sf;
         char Iseq[10];
       } Tr[10][128];  /* detecting tree */

main()
{
char ch,c;
clock_t time;

memzero(O,262144);
memzero(S,262144);
```

```
memzero(L,16384);
memzero(A,16);
memzero(D,65536);

printf("Test Generation for Stuck Faults Using\n");
printf("Boolean Difference and Transition Matrix Techniques.\n");
printf("This program stops whenever first detecting test is found\n");
printf("\n");
printf("Enter '1' ('0') for sequential (combinational) circuit\n");
i = 0;
while(i != 1)
  if((seq = getchar()) == '0'  || seq == '1')
  { if(seq=='1')
    { j = 0;
      printf("Enter '1' ('0') for Mealy (Moore) machine\n");
      while(j != 1)
        if((mealy = getchar()) == '0'  || mealy == '1')
          j=1;
    }
    i=1;
  }
printf("Enter # of inputs (<= 6), outputs (<= 8) and single faults (<=4):\n");
scanf("%u %u %u",&m,&n,&x);
if(seq=='1')
{ printf("Enter # of flip-flops (<= 8), and initial state (in decimal):\n");
  scanf("%u %u",&b,&Si);
}
else
  b = 0;
printf("Enter stuck values ('0' or '1') for %u faulty lines:\n",x);
for(i=0,stuckdec=0, ch=getchar(); i<x; i++, ch=getchar())
  if(ch=='0' || ch=='1')
  { stuck[i] = (ch - '0') ? '\1' : '\0';
    printf("X%d: s-a-%d, ",i+1,stuck[i]);
    if(stuck[i])
      stuckdec = stuckdec + pow[x-i-1];
  }
  else
    i--;
printf("\n");

/* Building reference matrix A */

for(i=0,c='\1',x2=pow[x]; i<x2; c='\1',i++)
{ for(j=0,sum=0; j<x; j++)
  { if(i>0)
```

```
    { A[i].binary[j] = c ¢ A[i-1].binary[j] ;
      c = (c && A[i-1].binary[j]) ? '\1' : '\0';
    }
    if(A[i].binary[j])
      sum = sum + pow[x-j-1];
  }
  A[i].value = sum;
}

/* Building O and L maps */

p = power(2,m+b);
b2 = power(2,b);
printf("Enter all functions in sum-of-product form, using '0','1','X'");
printf("; ' s' to finish.\n");
for(i=0; i<x; i++)
{ printf("Enter faulty line L%d function ",i+1);
  dec(L, i, m+b);
}

/* Boolean difference for output vector */

for(q=0,bdflag=sflag=found='\0'; (q<n && !found); bdflag='\0',q++)
{ printf("Enter output O%d function ",q+1);
  dec(O, q, x+m+b);
  if(mealy=='1' || seq=='0')
    printf("Single test(s) from BD of O%d = ",q+1);
  else
    printf("By RESET or CLEAR, difference at O%d = ",q+1);
  boolean(O, q);
  if( !found)
  { printf("not found ");
    if(seq=='1' && bdflag=='\1')
      printf("because of the initial state setting.\n");
    else
      printf("\n");
  }
}
if( !found)
{ printf("BD for outputs not found at all");
  if(seq=='0')
    printf(", fault is undetectable\n");
}

/* Working on next-state vector */
```

```
if(!found && seq=='1')
{   for(q=0,printf("\n") ; q<b; q++)
    { printf("Enter state S%d function ",q+1);
      dec(S, q, x+m+b);
    }
  if(trflag)
  { printf("This is Case II.  Let's try using detecting tree.\n");
    tree();
  }

  else
  { printf("This is Case I.  Let's try to find BD for next states.\n");
    for(q=0,sflag='\1'; (q<b && !bdflag); q++)
    { printf("BD for S%d = ",q+1);
      boolean(S, q);
      if( !bdflag)
        printf("not found\n");
      else
      { printf("found\nLet's try using detecting tree\n");
        tree();
      }
    }
    if( !bdflag)
    printf("BD for next states not found at all, fault is undetectable\n");
  }
}
printf("\nProcessor time used = %d milliseconds.\n",(time = clock()));

} /* end of main() */

/* Converting binary to decimal */

dec(addr, k, bit)
  char *addr;
  unsigned short k,bit;
{
 char ch,k2;
 unsigned short i,j,c,count;
 unsigned add[18];
 unsigned sum1;
 struct { unsigned sum;
          char c;
        } slot[154];

 k2 = pow[k];
 printf("\n (each term of %d bits in order: ",bit);
```

```
 if(bit == (x+m+b))
   printf("%d faulty lines, ",x);
 printf("%d inputs",m);
 if(seq=='1')
   printf(", and %d states):\n",b);
 else
   printf("):\n");
 while((ch=getchar()) != 's')
   if(ch=='0' || ch=='1' || ch=='X')
   { for(i=bit,c=0,sum=0,add[0]=0; i>0; i--,ch=getchar())
       switch(ch)
       { case '0' : break;
         case '1' :
           sum = sum + power(2,i-1);
           break;
         case 'X' :
           c++;
           add[c] = power(2,i-1) ;
           break;
         default :
           i++;
           break;
       }
     count = 1;
     slot[count].sum = sum;
     slot[count].c = '\0';
     while(count > 0)
     { sum1 = slot[count].sum;
       for(i=slot[count].c,count--; i<=c; i++)
       { sum1 += add[i];
         if(*(addr + sum1) < k2)
            *(addr + sum1) += k2;
         for(j=c; j>i+1; j--)
         { count++;
           slot[count].sum = sum1;
           slot[count].c = j;
         }
       }
     }
   }
}

/* To find the Boolean difference */

boolean(addr, r)
  char *addr;
```

```
  unsigned short r;
{
  unsigned short i,j,k,w;
  char bd,c,d,r2,B[4];

  r2 = pow[r];
  for(i=1,found='\0'; (i<x2 && !found); i++)
    for(j=0,A[0].status=0; (j<x2 && !found); j++)
      if(A[j].status != i)
        for(k=0; (k<p && !found); k++)
        {  c = j ¢ A[i].value;
           A[c].status = i;
           bd = ((*(addr+(p*j)+k) & r2) ¢ (*(addr+(p*c)+k) & r2));
           if(bd)
             for(w=0; (w<x && bd); w++)
             { if(A[i].binary[w])
                 B[w] = A[i].binary[w] ¢ stuck[w];
               else
                 B[w] = A[j].binary[x-1-w];
               d = (L[k] & pow[w]) ?  '\1' : '\0';
               if(B[w] != d)
                 bd = '\0';
             }
           if(bd)
           { trflag=bdflag='\1';
             if(!sflag)
             { if(seq=='1')
               { if(((k-Si) % b2) == 0 && !sflag)
                 { if(mealy=='1')
                     printf("%d with faulty output is ",(k-Si)/b2);
                   else
                     printf("found with faulty output is ");
                   found='\1';
                 }
               }
               else
               { printf("%d with faulty output is ",k);
                 found='\1';
               }
               if(found)
                 if(*(addr+(p*stuckdec)+k) & r2)
                   printf("1\n            ");
                 else
                   printf("0\n            ");
             }
           }
```

```
        }
} /* end of boolean */

/* Building Detecting Tree */

tree()
{
unsigned level;

printf("Enter number of maximum level of detecting tree (<=10):\n");
scanf("%u", &level);
for(i=0,sum=0;i<x;i++)
 if(stuck[i])
  sum=sum+pow[x-i-1];
for(w=0,j=0,m2=power(2,m),q=p*sum; j<m2; j++)
for(i=0; i<b2; i++,w++)
{ for(k=0,sum=0; k<n; k++)
   if(O[q+w] & pow[k])
    sum = sum + pow[n-k-1];
  T[i][j].Of = sum;
  for(k=0,sum=0; k<b; k++)
   if(S[q+w] & pow[k])
    sum = sum + pow[b-k-1];
  T[i][j].Sf = sum;/*end of faulty*/
  for(k=0,sum=0; k<x; k++)
   if(L[w] & pow[k])
    sum=sum+pow[x-k-1];
  for(k=0,y=0; k<n; k++)
   if(O[(p*sum)+w] & pow[k])
    y=y+pow[n-k-1];
  T[i][j].Oo=y;
  for(k=0,z=0; k<b; k++)
   if(S[(p*sum)+w] & pow[k])
    z=z+pow[b-k-1];
  T[i][j].So=z;
}
printf("The initial state is %d\n",Si);

/* To build the D matrix */

n2=pow[n];
for(j=0,detpair=0; j<m2; j++)
  for(i=0; i<b2; i++)
  {
   if(!(D[i][i].status))
    if(T[i][j].Oo != T[i][j].Of)
```

```
    { D[i][i].status = '*';
      D[i][i].I = j;
      D[i][i].Oo = T[i][j].Oo;
      D[i][i].Of = T[i][j].Of;
      detpair++;
    }
    for(k=i+1; k<b2; k++)
     {
      if(!(D[i][k].status))
       if(T[i][j].Oo != T[k][j].Of)
       { D[i][k].status = '*';
         D[i][k].I = j;
         D[i][k].Oo = T[i][j].Oo;
         D[i][k].Of = T[k][j].Of;
         detpair++;
       }
       if(!(D[k][i].status))
        if(T[i][j].Of != T[k][j].Oo)
       { D[k][i].status = '*';
         D[k][i].I = j;
         D[k][i].Oo = T[k][j].Oo;
         D[k][i].Of = T[i][j].Of;
         detpair++;
       }
     }
   }
if(D[Si][Si].status == '\0')
  D[Si][Si].status = '=';

if(!(detpair))
  printf("The fault is undetectable since there is no detecting pair.\n");
else
{ /* to build detecting tree */
  Tr[0][0].level=0;
  Tr[0][0].status= (D[Si][Si].status == '*') ?  '*' : '$';
  Tr[0][0].So= Tr[0][0].Sf= Si;
  for(i=0,nc=0,pc=1 ; (i<(level-1)) && (!found) && pc; i++,nc=0)
  { for(j=0; (j<pc) && (!found); j++)
     switch(Tr[i][j].status)  /* check current node status */
     { case '*':  /* initial pair is a detecting pair.. */
        found = '\1';  /*solution found ==> test length = 1 */
        printf("Detecting pair found, testing sequence(s) of length 1 :\n");
        printf("%d  with fault-free & faulty output: %d, %d\n",
                  D[Si][Si].I,D[Si][Si].Oo,D[Si][Si].Of);
        break;
       case '$':  /* generating more nodes */
```

```
        for(k=0; (k<m2) && (!found); k++)
        { Tr[i+1][j].level= i+1;
          y=T[Tr[i][j].So][k].So;  /* next generated  */
          z=T[Tr[i][j].Sf][k].Sf;  /* node */
          switch(D[y][z].status)  /* check next generated node status */
          { case '=':  /* it is a terminal node.. */
             break;    /* so, do not put in tree */
            case '\0':  /* it is a generating node */
             Tr[i+1][nc].status= '$'; /* update its status..*/
             D[y][z].status= '=';
             Tr[i+1][nc].So= y;  /*..and put it in tree..*/
             Tr[i+1][nc].Sf= z;
             for(q=0 ; q<i ; q++)   /*..and store input..*/
              Tr[i+1][nc].Iseq[q] = Tr[i][j].Iseq[q];   /*..sequence..*/
             Tr[i+1][nc].Iseq[q] = k;   /*..leading to this node*/
             nc++;   /* update # of nodes at next level */
             break;
            case '*':  /* it is a detecting node==>solution found */
             found= '\1';
             w = (mealy=='1') ?  i+2 : i+1;
             printf("Detecting pair found, testing sequence(s) of ");
             printf("length %d:\n",w);
             for(q=0 ; q < i ; q++)              /*..shortest..*/
               printf("%d ",Tr[i][j].Iseq[q]);   /*..test sequence*/
             printf("%d ",k);
             if(mealy=='1')
               printf("%d ",D[y][z].I);
             printf("with final fault-free & faulty outputs: %d, %d\n",
                      D[y][z].Oo,D[y][z].Of);
             break;
          }
        }
        break;
       case '=': break; /* current node is a terminal==>skip */
     }
     pc=nc;  /* load # of nodes at next level */
  }
   /* if a solution is not found */
   if(!found)
   { printf("The fault is undetectable since ");
     if(pc==0)
       printf("the detecting tree terminates.\n");
     else
       printf("the detecting tree reaches maximum %d levels.\n",level);
   }
}
```

```
} /* end of tree */

/* to calculate x to the yth power */

power(x,y)
unsigned short x,y;
{
  unsigned i,p;

  for(i=1,p=1; i<=y; i++)
   p=p*x;
  return(p);
}
```

Appendix C

PROGRAM OUTPUT LISTINGS FOR EXAMPLES 3 AND 4

```
Test Generation for Stuck Faults Using
Boolean Difference and Transition Matrix Techniques.
This program stops whenever first detecting test is found

Enter '1' ('0') for sequential (combinational) circuit
Enter '1' ('0') for Mealy (Moore) machine
Enter # of inputs (<= 6), outputs (<= 8) and single faults (<=4):
Enter # of flip-flops (<= 8), and initial state (in decimal):
Enter stuck values ('0' or '1') for 3 faulty lines:
X1: s-a-0, X2: s-a-0, X3: s-a-1,
Enter all functions in sum-of-product form, using '0','1','X'; ' s' to finish.
Enter faulty line L1 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter faulty line L2 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter faulty line L3 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter output O1 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O1 = not found
Enter output O2 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O2 = not found
Enter output O3 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O3 = not found
Enter output O4 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O4 = found with faulty output is 1

Processor time used = 224 milliseconds.
```

```
Test Generation for Stuck Faults Using
Boolean Difference and Transition Matrix Techniques.
This program stops whenever first detecting test is found

Enter '1' ('0') for sequential (combinational) circuit
Enter '1' ('0') for Mealy (Moore) machine
Enter # of inputs (<= 6), outputs (<= 8) and single faults (<=4):
Enter # of flip-flops (<= 8), and initial state (in decimal):
Enter stuck values ('0' or '1') for 3 faulty lines:
X1: s-a-0, X2: s-a-0, X3: s-a-0,
Enter all functions in sum-of-product form, using '0','1','X'; ' s' to finish.
Enter faulty line L1 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter faulty line L2 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter faulty line L3 function
 (each term of 10 bits in order: 2 inputs, and 8 states):
Enter output O1 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O1 = not found
Enter output O2 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O2 = not found
Enter output O3 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O3 = not found
Enter output O4 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O4 = not found because of the initial state
Enter output O5 function                                                  setting
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O5 = not found
Enter output O6 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O6 = not found
Enter output O7 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O7 = not found
Enter output O8 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
By RESET or CLEAR, difference at O8 = not found
BD for outputs not found at all
Enter state S1 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S2 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
```

```
Enter state S3 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S4 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S5 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S6 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S7 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
Enter state S8 function
 (each term of 13 bits in order: 3 faulty lines, 2 inputs, and 8 states):
This is Case II.  Let's try using detecting tree.
Enter number of maximum level of detecting tree (<=10):
The initial state is 0
Detecting pair found, testing sequence(s) of length 4:
3 0 0 0 with final fault-free & faulty outputs: 16, 32

Processor time used = 816 milliseconds.
```

```
Test Generation for Stuck Faults Using
Boolean Difference and Transition Matrix Techniques.
This program stops whenever first detecting test is found

Enter '1' ('0') for sequential (combinational) circuit
Enter '1' ('0') for Mealy (Moore) machine
Enter # of inputs (<= 6), outputs (<= 8) and single faults (<=4):
Enter # of flip-flops (<= 8), and initial state (in decimal):
Enter stuck values ('0' or '1') for 3 faulty lines:
X1: s-a-0, X2: s-a-0, X3: s-a-1,
Enter all functions in sum-of-product form, using '0','1','X'; ' s' to finish.
Enter faulty line L1 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter faulty line L2 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter faulty line L3 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter output O1 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O1 = not found
Enter output O2 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O2 = not found because of the initial state setting.
Enter output O3 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O3 = not found because of the initial state setting.
BD for outputs not found at all
Enter state S1 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S2 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S3 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S4 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S5 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S6 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
This is Case II.  Let's try using detecting tree.
Enter number of maximum level of detecting tree (<=10):
The initial state is 0
Detecting pair found, testing sequence(s) of length 8:
0 0 0 0 0 0 0 4 with final fault-free & faulty outputs: 6, 5

Processor time used = 2836 milliseconds.
```

```
Test Generation for Stuck Faults Using
Boolean Difference and Transition Matrix Techniques.
This program stops whenever first detecting test is found

Enter '1' ('0') for sequential (combinational) circuit
Enter '1' ('0') for Mealy (Moore) machine
Enter # of inputs (<= 6), outputs (<= 8) and single faults (<=4):
Enter # of flip-flops (<= 8), and initial state (in decimal):
Enter stuck values ('0' or '1') for 3 faulty lines:
X1: s-a-1, X2: s-a-0, X3: s-a-0,
Enter all functions in sum-of-product form, using '0','1','X'; ' s' to finish.
Enter faulty line L1 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter faulty line L2 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter faulty line L3 function
 (each term of 13 bits in order: 7 inputs, and 6 states):
Enter output O1 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O1 = not found
Enter output O2 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O2 = not found because of the initial state setting.
Enter output O3 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Single test(s) from BD of O3 = not found because of the initial state setting.
BD for outputs not found at all
Enter state S1 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S2 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S3 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S4 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S5 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
Enter state S6 function
 (each term of 16 bits in order: 3 faulty lines, 7 inputs, and 6 states):
This is Case II.  Let's try using detecting tree.
Enter number of maximum level of detecting tree (<=10):
The initial state is 0
Detecting pair found, testing sequence(s) of length 4:
0 0 0 8 with final fault-free & faulty outputs: 6, 5

Processor time used = 2830 milliseconds.
```