REPORT R-650 JUNE, 1974 UILU-ENG 74-2215

CSL COORDINATED SCIENCE LABORATORY

MULTIPLE FAULT DIAGNOSIS IN COMBINATIONAL NETWORKS

CHARLES WEI-YUAN CHA

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

MULTIPLE FAULT DIAGNOSIS IN COMBINATIONAL NETWORKS

by

CHARLES WEI-YUAN CHA

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under contract DAAB07-72-C-0259.

MULTIPLE FAULT DIAGNOSIS IN COMBINATIONAL NETWORKS

BY

CHARLES WEI-YUAN CHA

B.S., New Asia College, 1968
M.S., University of Illinois, 1970

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1974

Thesis Advisor: Professor Gernot A. Metze

Urbana, Illinois

MULTIPLE FAULT DIAGNOSIS IN COMBINATIONAL NETWORKS

Charles Wei-yuan Cha, Ph.D.
Coordinated Science Laboratory and
Department of Computer Science
University of Illinois at Urbana-Champaign, 1974

A new concept, the prime fault, is introduced for the study of multiple fault diagnosis in combinational logic networks. It is shown that every multiple fault in a network can be represented by a funcationally equivalent fault with prime faults as its only components. The use of prime faults greatly simplifies multiple fault analysis and test generation.

Also, masking faults for a given fault under a given test are defined. The conditions under which fault masking can occur are examined in detail. These conditions lead to an efficient method for generating a locally optimal test for a given prime fault.

With the above two new concepts, an efficient algorithm to generate a multiple fault detection test set for any combinational circuit has been developed. It not only will generate the multiple fault detection test set for an irredundant network; it also pinpoints any redundancy in a redundant circuit.

Bounds on the number of tests in the multiple fault detection test set are also found. The upper bound is the number of prime faults. This leads to a design principle, that is, a circuit should be designed with a minimum number of fanouts in order to yield a minimum number of tests in the multiple fault detection test set. A network designed according to this principle can be easily diagnosed.

ACKNOWLEDGMENT

The author is greatly indebted to his advisor, Professor Gernot Metze, for his inspiring guidance, priceless discussions and constant encouragement in the course of the research presented here.

The author wishes to thank the members of the Digital Systems Group of the Coordinated Science Laboratory and particularly Professor E. S. Davidson, Mr. James Smith, and Mr. Trevor Mudge for many useful discussions.

The support extended to the author during the past two years by the Coordinated Science Laboratory of the University of Illinois is gratefully acknowledged. The assistance of Mrs. Martha Meade and Mrs. Rose Harris in the typing of the manuscript is also greatly appreciated.

Finally, the author is very grateful to his wife Alice and to his parents, Chi-fu and Kwok-Ying Cha, for their love and encouragement throughout all of his school years.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER 1

INTRODUCTION

1.1 Reasons for Studying Fault Diagnosis

"To err is human."

The same applies to all machines including digital computers. Machines will malfunction due to various causes, like component disintegration, wrong connections, power failures, and so on. As years go by, digital computers are getting larger faster and more complex. Nowadays, computers are playing a role almost in every aspect of life---managing paychecks, checking identities, controlling production, monitoring air traffic, and so on. This is just the beginning. The extent of human reliance upon these huge machines has almost become a matter of life and death. A malfunction in a digital computer may cause a great deal of damage, expense or even loss of life. Therefore, it is very important that computers run properly, and this importance will increase considerably with time.

What is fault tolerant computing? It is the ability to execute specified algorithms correctly, regardless of hardware failures and software errors. Basically, the technology of fault tolerant computing encompasses theory and techniques of fault and error detection and correction, modeling, analysis, synthesis, and architecture of fault tolerant systems and their evaluation. Thus the scope of fault tolerant computing is broad. This thesis is mainly concerned with fault diagnosis. A fault diagnostician is concerned with the detection and location of

faults within a digital system. A fault in a circuit is detected if we apply some test to the inputs of the circuit and the outputs of the circuit differ from the correct outputs under the same test.

Digital systems have a large number of digital circuits, which are constructed from two basic circuit types, namely combinational circuits and sequential circuits. A combinational circuit is the more basic logical circuit, and sequential circuits are composed of combinational circuits plus memory. The discussion in this thesis is confined to combinational circuits, unless specified otherwise. In order to diagnose a circuit, one should be able to generate a set of tests. One naive way to generate a test set is to use all the possible input patterns of the circuit. Obviously, this will detect all the detectable faults in the circuit. But, the number of tests will exponentially increase as the number of inputs increases. This effectively excludes the exhaustive method of diagnosis in most cases, except for very small circuits. A good diagnostician should be able to generate an adequate set of tests for a given system and meet the following criteria.

(1) The test set should be complete. It should detect all the faults under consideration.

(2) The amount of computation required to generate the test set should not be excessive.

(3) The size of the test set should be small.

Until now, most algorithms for generating test sets have been concerned with single faults only. They assume the circuit cannot have more than one fault at any given time. This assumption is justifiable

only if testing is frequent enough so that the probability of the occurrence of more than one fault during the interval between tests is negligibly small. The probability of the occurrence of a single physical fault which produces several simultaneous logical faults is also assumed to be negligibly small. These two assumptions are not practical in most cases. For example, the single fault assumption may not be valid for the initial check-out of a circuit. Therefore, multiple fault analysis is justified. The main reason why multiple faults have thus far escaped thorough analysis is the large number of possible multiple faults. There are $3^n - 1$ multiple faults compared to only 2n single faults for a circuit with n lines. For example, a circuit with 40 lines has approximately 10^{19} possible multiple faults, but only 80 single faults.

1.2 Brief Literature Review

The most straightforward method for generating a test set for a combinational circuit is the use of a fault table. Each test is listed as a row heading and each fault as a column heading. For each test, there is an indication, in the table, of the faults it detects. Then, a minimal set of tests to detect all the faults can be found in the same way as a set of prime implicants is selected to cover all the minterms of a Boolean function. To guarantee minimality, all possible test patterns must appear in the fault table. Therefore, the application of this method must be limited to very small circuits. Kautz [1], Poage and McCluskey [2], Chang [3] and Powell [4] all describe this method.

Armstrong [5] and Sellers et al. [6] pointed out that by using the so-called Boolean difference, one can find tests for a given fault. However, this method requires a large amount of computation. Hence, it is not practical in most cases.

The method of using sensitized paths, to propagate the fault indication from the fault site to a primary output is very useful. This is the basis of many test generating procedures. Roth [7] formalized this concept and incorporated it into his D-algorithm to generate a test for a given single fault. Although the D-algorithm has its drawbacks, many methods which have been found are modifications of it.

Poage [8] proposed a method for writing the output expression of a digital network that incorporates the network structure. From that expression, a single fault detection test set can be derived by a fault table and a multiple fault detection test set can be derived by first finding some 'critical columns'. This is a very beautiful theoretical result. Unfortunately, practical usage of this method is limited as it takes too much time and effort for even a small circuit.

Schertz [9] proposed the concept of fault collapsing. He combines those faults which are inherently indistinguishable into equivalence classes. Then, the fault behavior of the circuit can be completely characterized by the fault classes. This approach leads to a tremendous reduction in the amount of work required. Schertz and Metze [10] also develop some design methods that simplify the method of detection of multiple faults in certain combinational networks.

Hayes [11] has studied normal NAND networks and examined the

conditions under which fault masking may occur. His study led to ways of determining if a set of tests is a multiple fault test set.

Yau and Tang [12] proposed to generate multiple fault test sets based on the method of Boolean differences. But, their method does not apply to all networks, and for certain multiple faults, reverts to Poage's method, which is known to be impractical.

Bossen and Hong [13] find check points in a circuit to characterize multiple faults. The method they derive is not very complex and computation is much simpler than Poage's method. However, the test set they derive is much larger than the optimal test set.

All the above existing methods are not satisfactory for finding multiple fault test sets for general combinational circuits. But, they have built a good foundation upon which a satisfactory method can be constructed. It is the goal of this thesis to generate a nearly optimal multiple fault test set efficiently for any combinational circuit.

1.3 Outline of the Thesis

In Chapter 1, an introduction as to the importance of multiple fault diagnosis is given. Also, the literature on other efforts in this area is reviewed briefly. In Chapter 2, some essential definitions are given. In Chapter 3, the main algorithm for generating a multiple fault detection test set for any combinational network is presented. Some examples are then given, and a comparison is made with other existing algorithms. In Chapter 4, some important theorems concerning the algorithm are proved. In Chapter 5, a new design rule is given which

tends to reduce the maintenance cost. Finally, conclusions and some topics for future research are listed.

CHAPTER 2

BASIC TERMS IN FAULT DIAGNOSIS

2.1 Introduction

In this chapter, various terms such as input test patterns, u-tests, e-tests, co-inputs, kernels or canonical fault representations, fan-out networks, masking, and redundancy, which are needed to understand the following chapters, will be defined.

2.2 A Fault Model and Its Representation

The logical model of a switching circuit under consideration is composed of AND, OR, NAND, NOR and NOT gates together with primary input lines, primary output lines, and intermediate lines which connect gates. A fault is any condition in a switching network which results in a change of the logical characteristics of that network. In most cases, the change of the logical characteristics can be logically characterized as certain lines of the network being stuck at logical 0 or at logical 1. This is the reason we are using the concept of the "stuck-at" fault.

Definition 2.1: A stuck-at fault component is some physical fault in a circuit which can be logically specified by saying that certain lines in the network are stuck at a constant logical value.

The line x stuck-at the logical value i will be denoted by x/i or x^i where i is 0 or 1.

The simplest stuck-at fault is one that affects only one

line and causes that line to be stuck at 0 or 1. We designate this as a single fault.

In contrast, a fault that causes more than one line to be stuck at a logical value will be called a multiple fault.

Also, in this thesis faults are considered to be permanent faults; that is, they do not disappear or change their nature during testing.

2.3 Network Structure

There are two different kinds of network structure of interest here. One is the tree structure and the other is the fan-out structure.

Definition 2.2: A network is called a tree structure network if every primary input line and every gate output line in the network enters at most one gate.

An example is shown in Fig. 2.1 a.

Definition 2.3: A network is called a fan-out network if at least one line in the network enters more than one gate.

An example of a fan-out network is shown in Fig. 2.1 b.

A tree structure network is also called a fan-out-free network. A special class of tree network is a restricted fan-out-free network which is shown in Fig. 4.3. Schertz and Metze [10] have proved that any single fault test set (SFTS) for the restricted fan-out-free network, where primary input fan-out is allowed, is also a multiple fault test set (MFTS). A simple proof of this result will be given in Chapter 4 of this thesis.

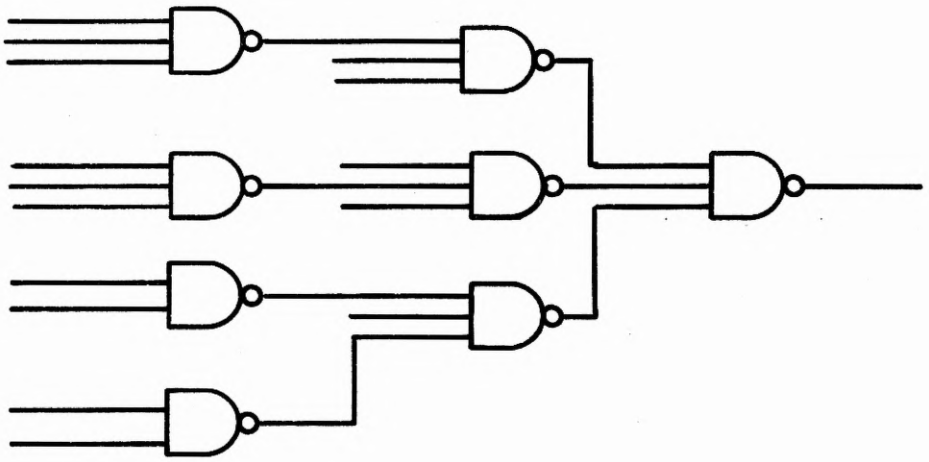


Figure 2.1 a) A Fan-out Free Network

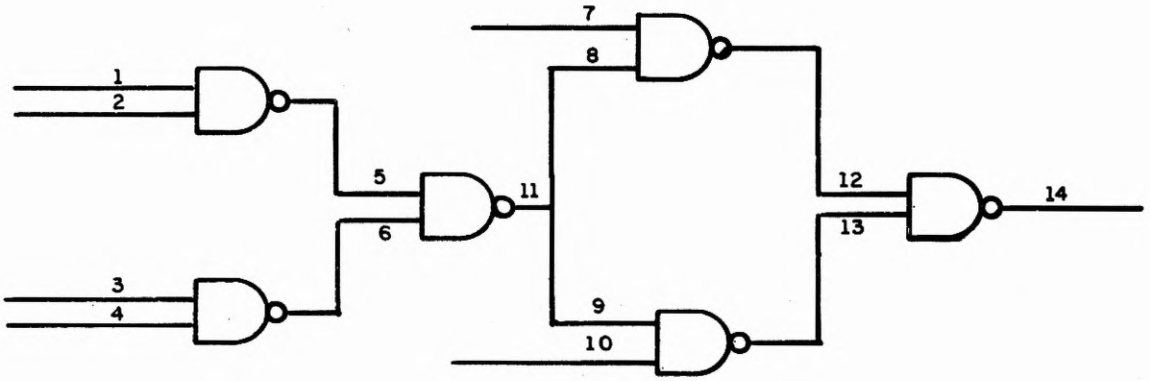


Figure 2.1 b) A Fan-out Network

In a fan-out network, the lines which come from the same stem are called branch lines of the fan-out stem. For example, lines 8 and 9 are fan-out branch lines of the stem line 11 in Fig. 2.1 b.

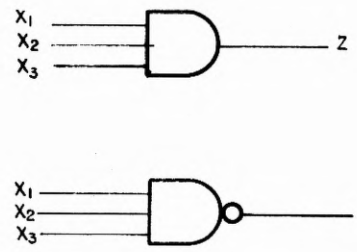
If two or more branch lines of a fan-out point reconverge at a later stage, as lines 8 and 9 do in Fig. 2.1 b, the network is called a reconvergent fan-out network. On the other hand, if fan-out branch lines never reconverge, the network is called a non-reconvergent fan-out network. It is obvious that non-reconvergent fan-out can occur only in multiple output networks. In a single output network, any fan-out is a reconvergent fan-out.

2.4 Input Test Pattern

For any gate with inputs $x_1, x_2, \ldots, x_n$, there are 2^n different input patterns. In test generation, the input patterns are very important. Hence, we classify the 2^n input test patterns into different categories.

Definition 2.4: For AND and NAND gates, if all the inputs are 1, we call this a u-test input pattern; if all the inputs are 1 except one input which is 0, we call this an e-test input pattern; if more than one input line is 0, we call this an m-test input pattern. An m-test input pattern for which all input lines are 0 is called $\bar{u}$-test pattern since its inputs are the complements of the u-test pattern. For OR and NOR gates, 0's and 1's are interchanged in the above.

Fig. 2.2 shows examples of all the different input test patterns for gates with three input lines.

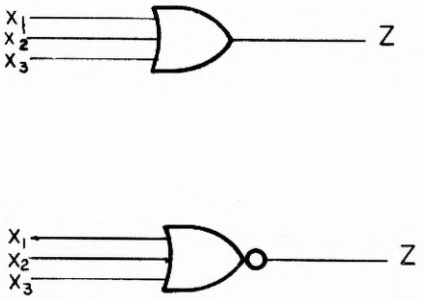


$U = (1,1,1)$

$\bar{U} = (0,0,0)$

$e_1 = (0,1,1)$
$e_2 = (1,0,1)$
$e_3 = (1,1,0)$
} e-TEST

$m_1 = (1,0,0)$
$m_2 = (0,1,0)$
$m_3 = (0,0,1)$
} m-TEST



$U = (0,0,0)$

$\bar{U} = (1,1,1)$

$e_1 = (1,0,0)$
$e_2 = (0,1,0)$
$e_3 = (0,0,1)$
} e-TEST

$m_1 = (0,1,1)$
$m_2 = (1,0,1)$
$m_3 = (1,1,0)$
} m-TEST

Figure 2.2 Various Input Test Patterns

Definition 2.5: For any gate G with inputs $x_1, x_2, \ldots, x_n$, the inputs $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ of G are called co-inputs of x_i.

0 is a dominating input signal for AND and NAND gates, i.e. if any input is 0, then the output will be 0 for AND or 1 for NAND, no matter what the values of the co-inputs of that input are. Similarly, 1 is a dominating input value for OR and NOR gates.

The e-input pattern is the only input pattern which will detect the line x_i stuck at 1 (0), where x_i has the applied value 0 (1) for AND (OR) or NAND (NOR) gates. The fault $x_i/1$ ($x_i/0$) can only be sensitized through the gate with this input pattern. If the test input for a gate G is an m-test pattern, there is more than one dominating signal on the input leads. In order for any sensitized path to pass through a gate G with an m-test input pattern, the test must sensitize all the input lines of G that have dominating values on them. In other words, the gate with an m-test pattern is less sensitive than under an e-test input pattern. Therefore, the more dominating values on the test inputs, the less sensitive the gate is. This is why the $\bar{u}$-test pattern, which has all dominating values on its input lines, is also called the optimal desensitizing input pattern. This is a very useful concept in test generation, as we shall see later.

2.5 Fault Relations

If two faults F_1 and F_2 in a network N are present at the same time, we will have a fault F_3 that is composed of F_1 and F_2. We call this composition on the set of faults 'concatenation'.

Definition 2.6: Let F_1 and F_2 be two stuck-at faults. The concatenation of F_1 and F_2 is defined by

$$F_3 = F_1 \cdot F_2$$

where F_3 is the union of the sets of component faults comprising F_1 and F_2, and the union of these component faults, in fact, constitutes a multiple stuck-at fault if they are defined.

As we know, a line cannot be stuck-at-1 and stuck-at-0 at the same time. Hence, if F_1 has the component i/1 and F_2 has the component i/0, then the concatenation $F_1 \cdot F_2$ is undefined.

The set of faults $\mathcal{F}$ in a network N under the operation of concatenation forms a commutative, regular semi-groupoid [14].

Definition 2.7: A fault F_1 in N dominates a fault F_2 in N if every test t of N that detects F_1 also detects F_2.

For a network N with the output function Z, we denote by Z_N the output function of the fault-free network N. The output function may change if the network is under some fault condition F; we denote by $Z_N[F]$ the output of network N under fault F.

Definition 2.8: Two faults F_1 and F_2 of the same network N are called functionally equivalent if $Z_N[F_1] = Z_N[F_2]$ for any input to N. We denote this by $F_1 \cong F_2$.

Theorem 2.1: Two faults F_1 and F_2 are functionally equivalent if and only if F_1 dominates F_2 and F_2 dominates F_1.

Proof: If $F_1 \cong F_2$, then $Z_N[F_1] = Z_N[F_2]$, so every test which detects F_1 will detect F_2 and vice versa. If F_1 dominates F_2, every test that detects F_1 also detects F_2. Similarly, if F_2 dominates F_1, every test

that detects F_2 also detects F_1. Also, all tests that do not detect F_1 do not detect F_2 and vice versa. Therefore, for any test, the output functions of the network N under Fault F_1 and F_2 are the same. By definition $F_1 \cong F_2$.

Q.E.D.

For a multiple fault, if the fault contains components both on inputs and outputs of some gate, then the output function of the network is affected by the fault on the output line only, as the input line fault is covered by the output line fault. This leads to the following definition.

Definition 2.9: A fault F_1 is an m-covering fault of another fault F_2 if the presence of F_2 together with F_1 affects the network N in the same way as F_1 does alone i.e. $Z_N[F_1] = Z_N[F_1 \cdot F_2]$. We denote this by $F_1 \supseteq F_2$.

Definition 2.10: Let F be a fault and F_1, F_2 be two subsets of F.

If (1) $F = F_1 \cdot F_2$

(2) $F_1 \supseteq F_2$, i.e. F_1 m-covers F_2.

(3) F_1 is the subset with fewest components which meet conditions (1) and (2), then, the fault F_1 is said to be a kernel of the fault F.

For any multiple fault F, if F_1 is the kernel of F, we will have $Z_N[F] = Z_N[F_1]$. Since F_1 m-covers $F_2 (= F - F_1)$, we can neglect the subfault F_2 completely and need to consider only those fault components in F_1. Therefore, we can always use the kernel F_1 of a multiple fault F to represent the fault F. We also call F_1 the canonical fault representation of F.

For fault m-covering, we also have the following theorem.

Theorem 2.2: If F_1 m-covers F_2 and F_2 also m-covers F_1, then $F_1 \tilde{=} F_2$.

Proof: F_1 m-covers $F_2 \Rightarrow Z_N[F_1] = Z_N[F_1F_2]$.

F_2 m-covers $F_1 \Rightarrow Z_N[F_2] = Z_N[F_2F_1]$.

Since the set of faults forms a commutative semi-groupoid, $F_1F_2 = F_2F_1$ if they are defined.

Therefore, $Z_N[F_1F_2] = Z_N[F_2F_1] \Rightarrow Z_N[F_1] = Z_N[F_2]$. So, by definition, $F_1 \tilde{=} F_2$.

Q.E.D.

Note--the converse of the above theorem is not true as we can see from the following example.

Example 2.1: For Fig. 2.3, let $F_1 = \{2/0,6/1\}$, $F_2 = \{1/1,5/0\}$, $F_3 = \{7/0\}$, $F_4 = \{1/1\}$. It is obvious that F_3 m-covers F_4 but neither F_3 dominates F_4 nor F_4 dominates F_3. This demonstrates the difference between fault dominance and m-covering. It is easily shown that $Z[F_1] = Z[F_2] = V \cdot W$. Therefore, $F_1 \tilde{=} F_2$. But $Z[F_1F_2] = Z[F_2F_1] = 0$, so neither F_1 m-covers F_2 nor F_2 m-covers F_1.

Also, it should be pointed out that functional equivalence is not a congruence relation. For example, $F_1 \tilde{=} F_2$ and $F_3 \tilde{=} F_4$ do not imply $F_1F_3 \tilde{=} F_2F_4$ as can be seen in the following example.

Example 2.2: In Fig. 2.4, let $F_1 = \{1/1,2/1\}$, $F_2 = \{3/1,4/1\}$, $F_3 = \{6/1\}$. $Z_N[F_1] \tilde{=} Z_N[F_2] = 1 \Rightarrow F_1 \tilde{=} F_2$. $F_3 \tilde{=} F_3$ since functional equivalence is an equivalence relation. But, $Z_N[F_1F_3] = 1$ and $Z_N[F_2F_3] = WX \Rightarrow F_1F_3 \not\tilde{=} F_2F_3$.

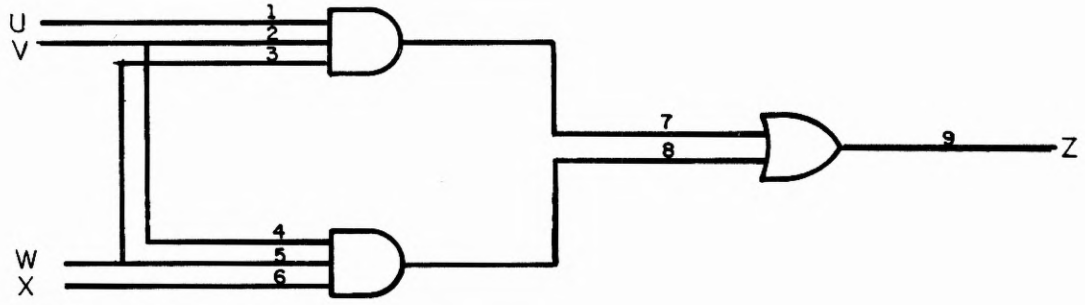


Figure 2.3

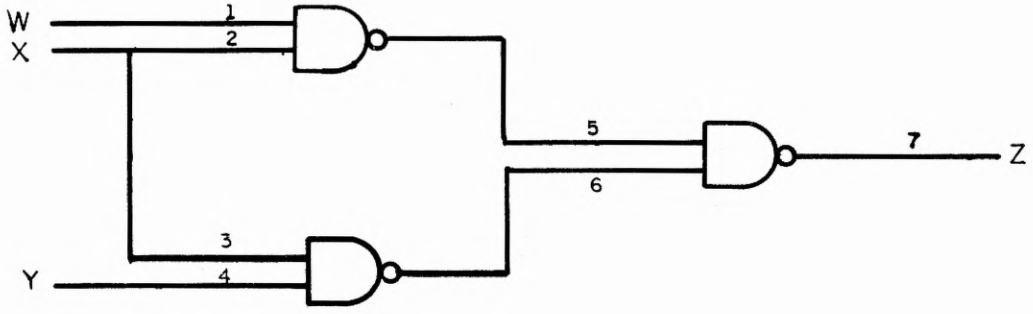


Figure 2.4

Since functional equivalence is an equivalence relation, it partitions the set of faults $\mathcal{F}$ into equivalence classes. Within any equivalence class, each individual fault affects the output of the network in the same way. Hence, in fault diagnosis we do not have to analyze every individual fault. Instead, we only need to analyze individual equivalence classes. As the number of equivalence classes is far less than the number of individual faults, this will reduce the fault analysis work tremendously. The next obvious question which arises is how all functional equivalence classes are found.

Schertz [15] has a very simple fault collapsing technique. It is listed in Table 2.2. The 1st and 3rd stages of fault collapsing form functional equivalence classes easily. The 2nd stage collapses dominant faults. Clegg and McCluskey [14] have defined S-structural, R-structural and functional equivalence. It can be shown that R-structural equivalence is resolved by Schertz's 1st and 3rd stages.

By using fault equivalence classes and kernel representations, we will always choose output lines stuck-at-1 as the fault representation of equivalence classes which consist of all input lines stuck-at-0 and output lines stuck-at-1 for NAND gates. Similarly, we will choose the corresponding stuck-at fault representation for other kinds of gates. Therefore, we will always choose stuck-at-1 faults to represent input line faults for NAND and AND gates and stuck-at-0 faults to represent input line faults for NOR and OR gates when analyzing faults other than faults on the primary outputs and fan-out stems. This immediately reduces the number of faults to be considered by almost a half. In

First Stage

Type	Description
fanout of 1	origin sa1 <=> destination sa1 origin sa0 <=> destination sa0
AND	any input sa0 <=> output sa0
OR	any input sa1 <=> output sa1
NAND	any input sa0 <=> output sa1
NOR	any input sa1 <=> output sa0
NOT	input sa1 <=> output sa0 input sa0 <=> output sa1

a.

Second Stage

Type	Description
non-reconvergent fanout	any destination sa1 ⇒ origin sa1 any destination sa0 ⇒ origin sa0
AND	any input sa1 ⇒ output sa1
OR	any input sa0 ⇒ output sa0
NAND	any input sa1 ⇒ output sa0
NOR	any input sa0 ⇒ output sa1

b.

Third Stage

Type	Description
fanout	origin sa1 <=> all destinations sa1 origin sa0 <=> all destinations sa0
AND	output sa1 <=> all inputs sa1
OR	output sa0 <=> all inputs sa0
NAND	output sa0 <=> all inputs sa1
NOR	output sa1 <=> all inputs sa0

c.

Table 2.2

Summary of Fault Collapsing Techniques

addition, the concatenation is always defined since we only choose one kind of stuck-at fault for every line except for the primary output line. NOT gates can be considered as either single-input NAND gates or single-input NOR gates. For uniformity, we consider NOT gates as single-input NAND gates.

The equivalence classes chosen by the above simplified method are good enough for most practical uses, even though the method will miss some equivalence classes, i.e. some equivalence classes may combine into one larger equivalence class as we can see from the following example.

Example 2.3: In Fig. 2.5 we have in a straightforward way

$C_1 = \{1/1\}$, $C_2 = \{2/1\}$, $C_3 = \{3/1\}$, $C_4 = \{4/1\}$, $C_5 = \{5/1\}$,
$C_6 = \{6/1\}$, $C_7 = \{1/0,5/0,7/1\}$, $C_8 = \{2/0,3/0,10/1\}$
$C_9 = \{4/0,6/0,8/1\}$, $C_{10} = \{7/0,8/0,9/1\}$, $C_{11} = \{9/0\}$,
$C_{12} = \{10/0\}$.

For single faults, there are 12 fault equivalence classes to be considered instead of 20 individual faults.

But, the classes $\{C_5,C_6,C_8\}$; $\{C_2,C_9\}$; $\{C_3,C_7\}$ and $\{C_1,C_4\}$ can be combined into single equivalence classes. Actually, we need to form only 7 equivalence classes for single fault analysis.

From the above example, we can see the number of combined equivalence classes is less than the number of equivalence classes that we had initially. Unfortunately, it is not easy to find the minimum number of equivalence classes for all single faults. The above example is a very special case. In most cases the number of equivalence classes

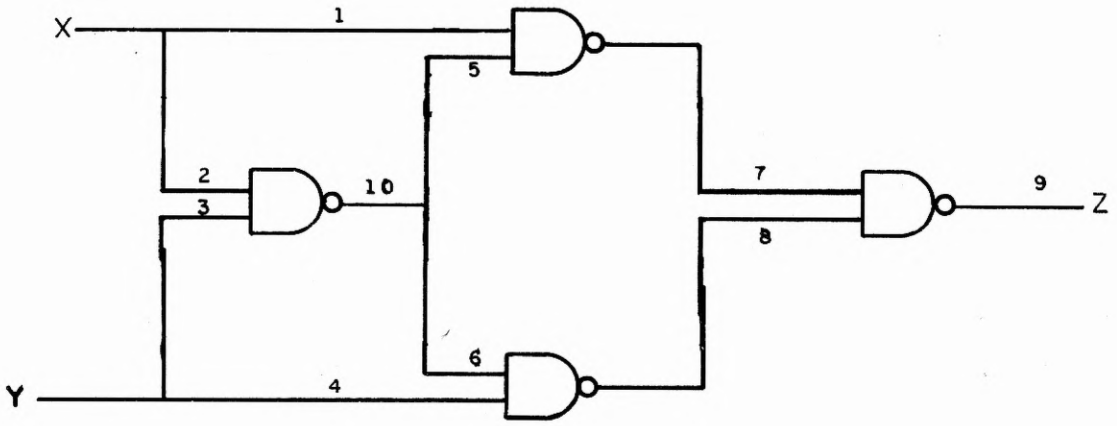


Figure 2.5

formed by the collapsing method is just slightly larger than the minimal number of equivalence classes. The remainder of this section will discuss some relations between functional equivalence classes and R-structural equivalence classes.

Clegg and McCluskey [14] have given necessary and sufficient conditions for two faults to be S-structurally equivalent and R-structurally equivalent. Unfortunately, they failed to give a necessary and sufficient condition for functionally equivalent faults and the functional equivalence relation is the most useful relation in single fault analysis. A necessary and sufficient condition for two faults to be functionally equivalent will be given here. However, Clegg and McCluskey have proved that two faults F_1 and F_2 may be functionally equivalent but not R-structurally equivalent in an irredundant network if and only if there exists at least one reconvergent fan-out path. Hence, to find functionally equivalent faults that are not R-structurally equivalent we have to concentrate on reconvergent fan-outs.

Theorem 2.3: Let F_1 and F_2 be two stuck-at faults in a network N. Then F_1 is functionally equivalent to F_2 if and only if either F_1 and F_2 are R-structurally equivalent or there is reconvergent fan-out in N such that the output at the reconvergent point of the reconvergent fan-out under the presence of F_1 is the same as it is under F_2.

Proof: Sufficiency--If F_1 and F_2 are R-structurally equivalent they are functionally equivalent. Now, if the outputs at the reconvergent point of the reconvergent fan-out are the same under F_1 and F_2, both F_1 and F_2 have to be stuck-at faults which occur before the reconvergence

point. We can disregard all the lines and gates before the reconvergent point since F_1 and F_2 will produce the same value at the reconvergent point. Consider the reconvergent point as a special input. Then the output of the network will be the same for F_1 and F_2, i.e. $Z_N[F_1] = Z_N[F_2]$. By definition 2.8 $F_1 \cong F_2$.

Necessity: If $Z_N[F_1] = Z_N[F_2]$, then if F_1 and F_2 are R-structurally equivalent, the proof is finished. So, assume F_1 is not R-structurally equivalent to F_2. By the theorem given by Clegg and McCluskey, the network must have a reconvergent fan-out, such that both F_1 and F_2 will affect the reconvergent fan-out. If at the reconvergence point of the reconvergent path the functional value in the presence of F_1 is different from the value in the presence of F_2, then since the reconvergent point is essential to the primary output, $Z_N[F_1]$ cannot be equal to $Z_N[F_2]$. This contradicts our assumption. Therefore, at the reconvergent point of the reconvergent path, the output must be the same under F_1 as under F_2.

Q.E.D.

The above theorem gives a criterion which determines if two faults are functionally equivalent. This will reduce the work required to form equivalence classes of single faults.

Although we have the necessary and sufficient condition for two faults to be functionally equivalent, the method suggested by the condition is not as easy to apply as the fault collapsing technique. Therefore, in most cases, we are content to use the simplified collapsing technique for equivalence classes of all single faults.

To find an SFTS for a circuit, we need to find all the equivalence classes of all single faults. Then, we only need to find a set of tests that will detect every equivalence class instead of individual faults. However, we have some difficulty in finding the minimum number of equivalence classes.

In deriving an MFTS, we don't have to find all the equivalence classes of single faults. We only need to find a small set of faults which can be used to model all the multiple faults. Therefore, for MFTS generation, we don't have the difficulty that we face in SFTS generation. Generating an MFTS is not only more natural, but is somewhat easier than generating an SFTS. We will discuss this property in the next chapter.

2.6 Masking Faults

A test t is said to detect a fault F in a network N if $Z_N \neq Z_N[F]$ under the test t.

Generally, a test t detects F and at the same time also detects many other faults. For the purpose of generating an MFTS, we are interested in knowing exactly which faults it detects and which faults it does not detect.

Definition 2.11: A fault F_2 is called a masking fault for F_1 under test t if the test t detects F_1, but does not detect $F_1 \cdot F_2$ and F_2 does not m-cover F_1.

For a given test t which detects F there may or may not exist masking faults. If there are masking faults, then we say t detects F

unconditionally. If there is a set of masking faults for F, denoted by S_F, we will say the test t detects F S_F-conditionally.

Example 2.4 will illustrate the above terms.

Example 2.4: In Fig. 2.6, the test $x_1x_2x_3x_4x_5x_6$ = 011011 detects $x_1/1$ unconditionally. But the test $x_1x_2x_3x_4x_5x_6$ = 011000 detects $x_1/1$ S_F-conditionally where $S_F = \{x_4/1\}$, since $x_4/1$ masks $x_1/1$ under this test.

From the above example, we can see that for a given fault F, different tests which detect F will detect different multiple faults which contain F as a component. This is a very important characteristic in deriving an MFTS. Under what conditions can a fault mask another fault? Hayes [16] stated some criteria in his thesis and this topic will be discussed more in the next chapter.

2.7 Redundancy

As pointed out in section 2.5 we are only considering stuck-at-1 faults on the inputs of AND and NAND gates and stuck-at-0 faults on the inputs of OR and NOR gates, because other input stuck-at faults are equivalent to their respective output stuck-at faults. Therefore, there is only one fault class which is associated with a given line; this class is called the representative fault class of that line. Representative fault classes will be implicitly used throughout this thesis.

The fault representative of a fault class is associated with a line. There is a strong relation between the lines and their representative faults, i.e. a stuck-at-1 (0) fault is associated with the inputs of AND (OR) and NAND (NOR) gates. We need the following definition.

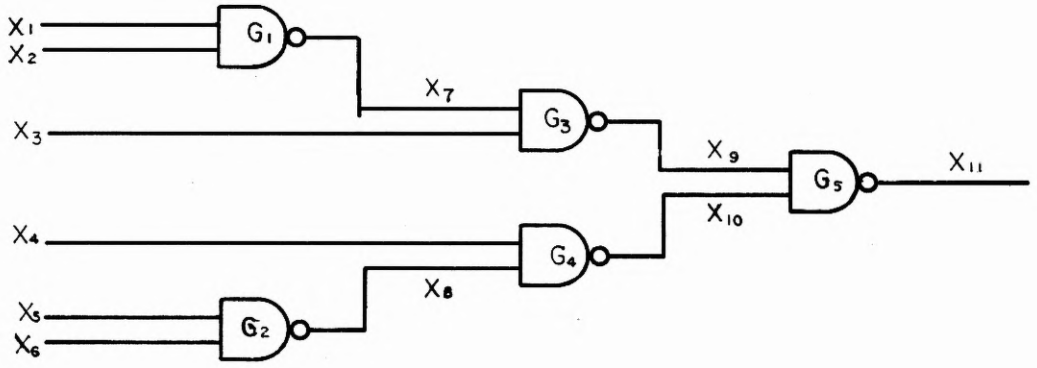


Figure 2.6

Definition 2.12: We define the fault associated with a set of lines as the fault which occurs when the lines are stuck-at their representative fault class value.

For certain networks, some lines can be stuck at the logical values 0 or 1 without affecting the output of the network, and these lines could therefore be removed from the network without affecting its operation. We will call these lines redundant lines and the circuit a redundant circuit, in the sense that these lines are unnecessary for the logic network.

Definition 2.13: A circuit is single-line redundant if there is some single line that can be removed from the circuit without affecting the logical behavior of the circuit. If a circuit is not single-line redundant we call it single-line irredundant.

Friedman [17] first pointed out that, in redundant circuits, a fault that is originally detectable may become undetectable due to faults on the redundant lines, which are undetectable. Hence, redundancy is very undesirable from a fault diagnosis viewpoint. Though some redundancy like Triple Modular Redundancy (TMR) and some redundant checking circuits are useful for producing higher reliability for a short length of time, unintentional logic design redundancies are a waste of money and are difficult to detect. Therefore, unintentional redundancy is very undesirable in logical design. This leads to an obvious question: how can one find all the redundancies in a network? Before we tackle this question, we give an equivalent, alternative definition of redundancy.

Definition 2.14: A set of lines is redundant if there does not exist a test to detect the fault which is associated with this set of lines.

The two definitions for redundancy are equivalent, since if a line is redundant, then, the output is not sensitive to the value on the redundant line, but, in order to detect the fault on this line, we have to sensitize the fault from that line to the output. If the output is not sensitive to the line, we are unable to detect the fault and vice versa.

With the above definition, we can determine if a circuit is single-line redundant or not by deriving an SFTS for the circuit. If there does not exist an SFTS for the circuit, then the circuit is single-line redundant; otherwise, it is single-line irredundant. Many methods exist for generating an SFTS. Therefore, the problem for detecting single-line redundancy has been solved.

At first glance, one may think that if a circuit is single-line irredundant, it will necessarily be an irredundant circuit. Unfortunately, this is not true. The circuit in Fig. 2.7 is single-line irredundant, but has a 4-line redundancy. Hence, to detect whether a circuit is redundant is a very difficult job. To verify that a circuit is multiple-line irredundant it is necessary to first check all single lines, then every pair, triple, and so on. 2^n possible redundancies must be checked. This is practically impossible for even a moderate circuit. So far the literature does not show a good method to detect all the redundancies in a circuit. A method which detects all redundancies in a circuit as a by-product of generating the MFTS for the circuit

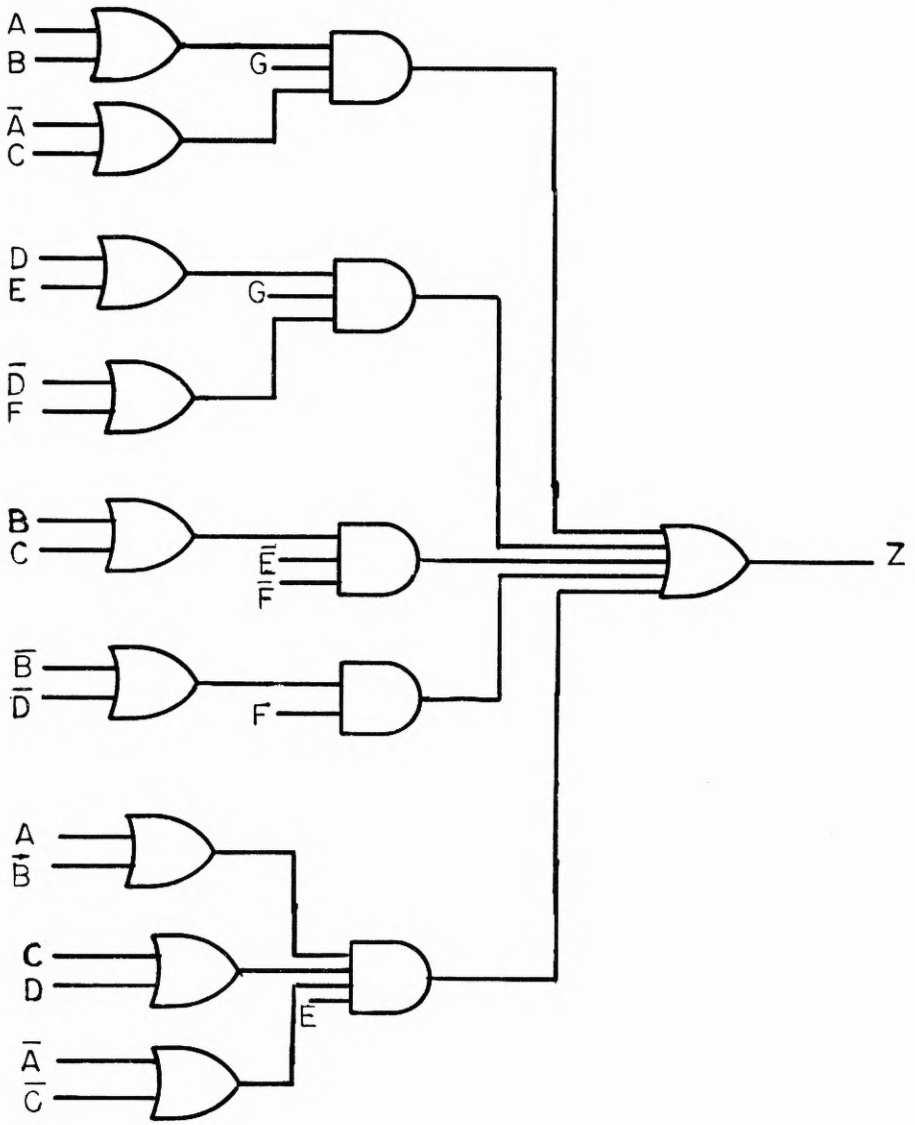

A
B
G
$\bar{A}$
C
D
E
G
$\bar{D}$
F
B
C
$\bar{E}$
$\bar{F}$
Z
$\bar{B}$
$\bar{D}$
F
A
$\bar{B}$
C
D
$\bar{E}$
$\bar{A}$
$\bar{C}$

Figure 2.7

will be presented.

The example in Fig. 2.7 shows a single-line irredundant circuit which is a four-line redundant circuit. Until now, the existence of a combinational circuit that is single-line irredundant but is two-line or three-line redundant has neither been proved nor disproved. However, for certain circuit configurations we have some theorems [10], [21] that are discussed in Chapter 4.

CHAPTER 3

AN ALGORITHM FOR GENERATING A MULTIPLE FAULT DETECTION TEST SET

3.1 Introduction

Procedures for generating an SFTS usually involve the use of the D-algorithm [7], Boolean difference [5,6] or some other form of path sensitization method [1,4,11,12]. Some procedures for the generation of MFDTS's* have appeared in the literature [2,8,9,11,12,13,25]. However, these procedures are essentially extensions of the standard single-fault approach which differ from it only in that the bookkeeping is considerably more tedious and they tend, therefore, to be very inefficient. As multiple fault detection becomes more important with the advance of large scale integrated circuits, a new look at multiple fault diagnosis is needed. In this chapter, an algorithm for MFDTS generation that is inherently efficient will be developed. First, prime faults are defined and a new path sensitizing technique for generating tests efficiently and for finding the corresponding masking faults is described. Then, the algorithm is summarized. Lastly, some examples to illustrate this algorithm and compare it with other methods are given.

3.2 Prime Faults

Freed from the single fault assumption, it becomes possible to represent all the faults in a network by a subset of all the single faults such that any fault in the network will be functionally equivalent to a fault with components in that subset of faults only. Therefore we

* MFDTS = Multiple fault detection test set.

define the set of prime faults as follows.

Definition 3.1: A subset of faults in a network N is called a set of prime faults if for any fault F in N, we can find a functionally equivalent fault F^1 whose components are in the set of prime faults only. In other words $Z(F) = Z(F^1)$. Any member of the set of prime faults will be called a prime fault.

Definition 3.2: A gate is called a prime gate if all the input lines of this gate are either primary input lines or branch lines of some fan-out points.

As we are concerned with multiple faults, from now on, whenever we refer to a fault, we mean either a single fault or a multiple fault, unless we specify otherwise. Also, we do not have to consider faults on fan-out stems, because a stuck-at-1 or stuck-at-0 fault at the fan-out stem is just equivalent to the multiple fault with every branch line of the fan-out stuck-at-1 or stuck-at-0. A procedure for finding a set of prime faults is presented as follows.

Procedure 1:

a) Pick all s-a-1 (s-a-0) faults on all the branch lines of all fan-outs that lead to NAND (NOR) or AND (OR) gates.

b) Pick all s-a-1 (s-a-0) faults on all the primary input lines that do not fan-out (if they do, they were already picked in step (a)) and that lead to NAND (NOR) or AND (OR) gates.

c) Pick the s-a-1 (s-a-0) fault on the output lines of prime gates which do not fan-out, where the prime gate is a NAND

(NOR) or OR (AND) gate.

We use PF to denote the set of prime faults generated by Procedure 1.

Theorem 3.1: PF forms a set of prime faults.

Proof: A network can have only two kinds of gates, namely, prime gates and non-prime gates. By Procedure 1 we have a representative for each fault equivalence class for prime gate input lines. For example, if the prime gate is a NAND gate and its output goes to another NAND gate, then we have s-a-1 faults for all the input lines of the prime gate by step (a) and (b). Also we have a s-a-1 fault for the output line of the prime gate by step (c), which is functionally equivalent to any input line of that prime gate s-a-0. For all other types of prime gates which lead to any different gate, the reader can verify for himself that we have chosen representatives for all the input lines of the prime gate s-a-1 and s-a-0.

Now, non-prime gates must have at least one input line that is neither a primary input line nor a branch line of some fan-out. For this line, we can trace back to either a primary input line or a branch line of some fan-out. Hence, for that line to be s-a-1 or s-a-0, we can find a functionally equivalent fault that has components in PF only. If the non-prime gate also has a primary input line or a branch line of some fan-out, it does not change anything.

As we know [13], any multiple fault can be traced back ultimately either to a primary input line or branch line of some fan-out. From Procedure 1, we have chosen equivalence classes which include all theses lines s-a-1 and s-a-0. Therefore, any multiple fault can

be represented by some fault or a combination of faults in PF only. By definition, this is a set of prime faults.

Q.E.D.

A line is called a prime line if it has a prime fault on it.

When we are generating the SFTS for a network, it is sufficient to pick one representative for each single fault equivalence class. In dealing with multiple faults, we want one representative for each multiple fault equivalence class. We need all the prime faults to make sure the test set will detect any multiple fault. Since functional equivalence is not a congruence relation as we pointed out in Example 2.2, we cannot delete any single prime fault even though it may be functionally equivalent to another prime fault.

Example 3.1: See Fig. 2.5 of Chapter 2. By Procedure 1, we will have 1/1, 2/1, 3/1, 4/1, 5/1, 6/1, 7/1, and 8/1 as members of the set PF for that circuit. Although we have $1/1 \cong 4/1$, $2/1 \cong 8/1$, $3/1 \cong 7/1$ and $5/1 \cong 6/1$, neither the set $S_1 = \{1/1,2/1,3/1,5/1\}$ nor the set $S_2 = \{4/1,6/1,7/1,8/1\}$ can be deleted from the set PF. We cannot delete 4/1, 6/1, 7/1 and 8/1 from the set PF, because the set $S_1 = \{1/1,2/1,3/1,5/1\}$ is not a complete set of prime faults. As we can see, the fault $F = \{6/1,7/1\}$ is not equivalent to the fault $F_1 = \{3/1,5/1\}$, because $Z(F) = Y$ and $Z(F_1) = \bar{X} + Y$. Therefore, S_1 is not sufficient as a set of prime faults. Similarly, S_2 is not sufficient.

From Procedure 1, we also have the following theorem.

Theorem 3.2: The number of prime faults in PF is equal to $N_1 + N_2 + G_1 - G_2$ where N_1 is the number of primary input lines which do not fan-out,

N_2 is the total number of all branch lines of all fan-out points, G_1 is the total number of prime gates, and G_2 is the number of prime gates whose output lines fan out. The proof is immediate.

Now, we know how to find a set of prime faults of a network. If we can find a set of tests that will detect all the prime faults and their combinations, then we have an MFDTS. In the next section, we will describe an efficient method for generating a test for a given prime fault together with its corresponding masking faults.

3.3 Test Generation and Fault Masking

To find a test for a given prime fault, one can use Boolean difference, path sensitization or some other technique. We tend to use the path sensitization concept for generating the test, because we feel this is the most natural and direct method. However, the path sensitization we will use is different from the widely known D-algorithm. In the D-algorithm, one has to choose one sensitized path at a time and desensitize all others. If a given prime fault needs more than one sensitized path in order to be detectable, one has to exhaust all the single sensitized paths first. Then, one has to sensitize every pair of sensitized paths and so on until a test is found. Therefore, the method is very time-consuming. In order to avoid the above difficulty, we have the following theorem.

Theorem 3.3: In generating a test for a given fault, if more than one path must be sensitized in order to detect that fault, then one can select one path to be sensitized arbitrarily and the chosen sensitized

path will automatically sensitize other paths as needed. That is to say, one never has to choose more than one sensitized path deliberately.

Proof: Suppose in a circuit N, a given fault f requires more than one path to be sensitized in order to detect f. Now, if we choose one sensitized path and deliberately desensitize all the other paths, we will get a contradiction on some line, because sensitizing that one path alone will not allow us to detect f. Therefore, some of the lines we deliberately assigned desensitizing values to in order to desensitize them are dependent on the values of the chosen sensitized path. Let us call the contradictory line a "kill" line.

Now, after we pick a path, in order to assure that the path will be sensitized, we assign all the values on the sensitized path and its co-inputs except those which are the inputs of reconvergent fan-out gates. Now, because of these assigned values on the selected sensitized path the kill line will be forced to have a sensitized value on it. Therefore, the gate with the kill line as one of its inputs will also be sensitized. This will sensitize the other path. Therefore, one never is required to sensitize more than one path deliberately. If another path is needed, it will be automatically sensitized.

Q.E.D.

It is well known that more than one path must be sensitized in order to detect certain faults. Let us use the Schneider example [7] to illustrate Theorem 3.3.

Example 3.2: Fig. 3.1 is the Schneider example. We have to sensitize two paths in order to detect line 20 s-a-0. We use the D-algorithm,

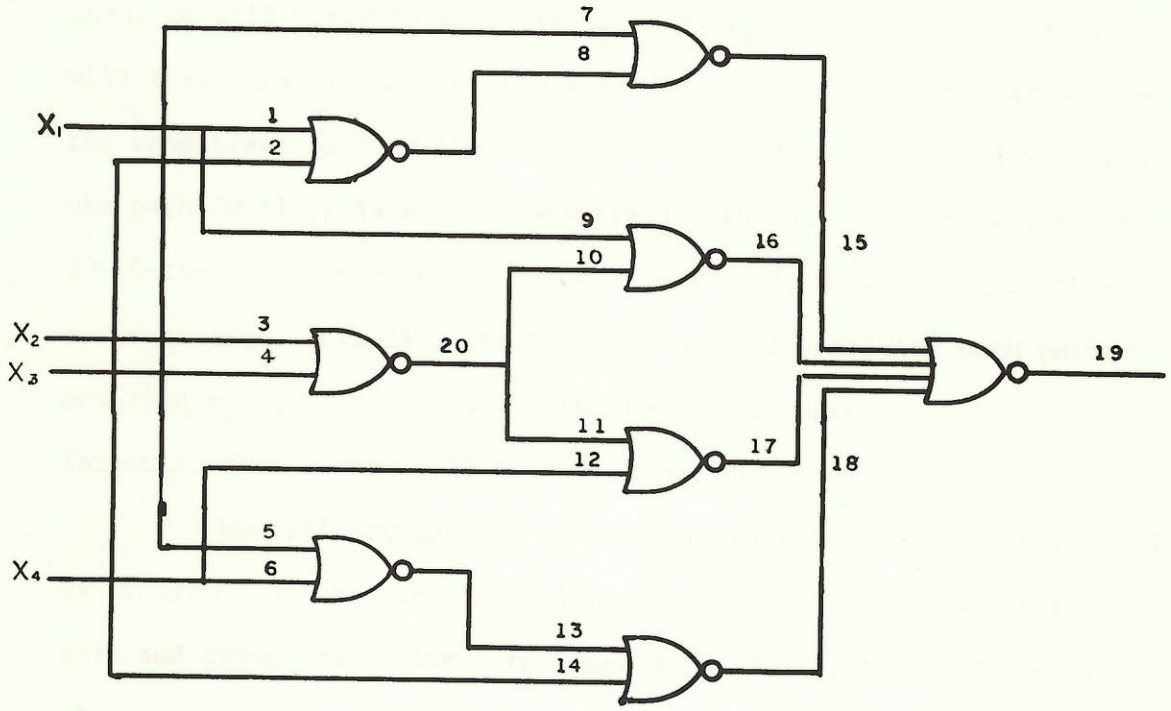


Figure 3.1 Schneider Example

and choose line 20-10-16-19 as a sensitized path first. Therefore, we must apply the value 0 on lines 3, 4, 9, 15, 17, 18 and the value 1 on line 12 to desensitize the path consisting of lines 20-11-17-19. Then, we will generate the test $x_1\ x_2\ x_3\ x_4 = 0001$, but this test will force line 18 to have the value 1. This is a contradiction and the sensitized path 20-10-16-19 must be abandoned. Now, if we choose the path 20-11-17-19 as the sensitized path and desensitize the path 20-10-16-19 by applying the value 1 on line 9, we will get another contradiction. Finally, the D-algorithm will sensitize both paths and find $x_1\ x_2\ x_3\ x_4 = 0000$ as the test. For networks with more fan-outs, this method will be very time-consuming.

We will now use the new sensitized path technique suggested by Theorem 3.3. We can choose lines 20-10-16-19 as the sensitized path and assign the value 0 to lines 3, 4, 9, 15, 18 and leave line 17 unassigned for the moment as it is an input to a reconvergent fan-out gate. Now, since line 18 has the value 0, we must assign line 13 the value 1 because line 14 has the value 0 already due to the value 0 on line 4. In order to have the value 1 on line 13, we have to assign the value 0 to line 5 and line 6. Now, since line 6 has the value 0 so does x_4 and so does line 12. This will sensitize the path 20-11-17-19 automatically without deliberation. Therefore, we get the test 0000 easily. Note here that line 12 is a so-called kill line.

Therefore, in the test generation step, we will use this new path sensitization technique to generate the test; this will avoid the tedious trial and error steps in the D-algorithm.

For a given fault, after generating the partial test that sensitizes a path and will detect the given fault, we have to complete the test. This is the consistency step. Since we want an MFDTS, we must generate a test set that will not only detect all the prime faults, but will also detect all their combinations. Therefore, it is very important to find the set of masking faults for a given fault f under a generated test. Any test that detects a given fault will sensitize the fault from the fault site to the primary output line by at least one sensitized path. Along the sensitized path, we notice that the gates can be divided into two categories.

Definition 3.3: Along a sensitized path, a gate will be called a type 1 gate if it has an e-vector input pattern and the desensitizing value is on the sensitized path. A gate on the sensitized path will be called a type 2 gate if it has a u-vector input pattern. If the fault requires that more than one sensitized path be used to detect it, then at a reconvergent fan-out gate G, more than one input line is on a sensitized path. If all the inputs on sensitized paths have desensitizing values, gate G is called a type 1a gate. If they all have sensitizing values, gate G is called a type 2a gate. Note - the input lines to gate G, which are on sensitized paths, must all have desensitizing values or must all have sensitizing values, otherwise the reconvergent fan-out gate is not sensitized, which is contradictory. Also, we do not consider the gate at the reconvergent fan-out as being on the path of the reconvergent fan-out.

A type 1 subnetwork is a subnetwork that feeds a type 1 gate or type 1a gate and which contains no line that belongs to a sensitized path. A type 2 subnetwork is a subnetwork that feeds a type 2 gate or a type 2a gate and which contains no line that belongs to a sensitized path.

It is easy to show that masking faults in a circuit for a given fault can only mask at type 1 gates or type 1a gates. Therefore, we should apply $\bar{u}$-vectors, or optimally desensitizing vectors, to all the gates of a type 1 subnetwork whenever possible. By doing this, we can minimize the number of masking faults for a given fault. On the other hand no masking can occur at a type 2 gate or a type 2a gate. Since they have sensitizing values on all their inputs, they can be part of any sensitized path through any one of their inputs. Therefore, we should try to assign an e-vector input test pattern to all the gates of a type 2 subnetwork whenever possible. By doing this, the test we generate will detect some additional faults. We should always try to let the additional detectable fault be a prime fault. This may reduce the size of the generated MFTS. Now, we are ready to present a procedure which generates a test for a given fault and gives its associated masking faults.

Procedure 2:

i) For a given prime fault, prime line j s-a-1 (s-a-0), apply the value 0 (1) to the prime line j. Also, assign sensitizing values to all of the co-input lines of line j.

ii) Choose a path from line j to the primary output line as a sensitized path. Assign sensitizing values to all the co-input lines of the sensitized path except those input lines at reconvergent fan-out gates.

iii) Assign values to all the lines which are determined by (i) and (ii). For example, if in step (i) and (ii) we have assigned the value 0 to the output of a NAND gate, then, this value 0 determines the values on all the input lines of this NAND gate.

iv) Check lines at the reconvergent fan-out gates. If one is already defined due to step (iii) without causing any conflict, we have a type 1a gate or type 2a gate at the reconvergent fan-out gate. If one line is still undefined, assign the sensitizing value to it. If a line determined by step (iii) always desensitizes the gate at the reconvergent fan-out, then it will desensitize the path. Therefore, the fault cannot be detected by our chosen path. Choose another path as the sensitized path and go back to step (ii). If we have exhausted all the paths, the fault in question is undetectable. Exit.

v) This is the consistency step. All lines in the circuit either belong to the sensitized path or are co-inputs to lines on the sensitized path, or belong to type 1 subnetworks or type 2 subnetworks. Previous steps have

defined the values on all lines except some belonging to type 1 subnetworks or type 2 subnetworks. We start from the beginning of the sensitized path, namely the fault site, and assign values to all the lines of the circuit according to the type of subnetwork they belong to.

va) Assignment of values to type 1 subnetwork. Masking faults may exist in a type 1 subnetwork. Therefore, we should assign $\bar{u}$-vector input patterns to all gates in type 1 subnetworks whenever possible. Since this is not always possible, we must consider a number of cases.

Case I The type 1 gate which the type 1 subnetwork feeds is not on the path of reconvergent fan-out.

Case Ia If we can apply $\bar{u}$-vector input test patterns to all the gates in this type 1 subnetwork, there is no masking fault in this subnetwork.

Case Ib If we can not apply $\bar{u}$-vector input test patterns to all the gates in this type 1 subnetwork, there are faults that can be masking faults. However, if a fault happens to sensitize a path other than the original sensitized path, such that the newly formed sensitized path passes through a type 2 or a type 2a gate, which is closer to the output than the type 1 gate under consdieration, then that fault will not be a masking fault.

Case II The type 1 gate which the type 1 subnetwork feeds is on the path of reconvergent fan-out.

Case IIa The gate at the reconvergent fan-out is not a type 2a gate. If there is a line which needs to be defined, it has a fault that can be a masking fault, unless that fault will sensitize a path other than the original sensitized path such that the newly formed sensitized path passes through a type 2 or a type 2a gate which is closer to the output than the type 1 gate under consideration.

Case IIb The gate at the reconvergent fan-out is a type 2a gate. In this case, a masking fault is formed as a combination of one fault for each sensitized path of the reconvergent fan-out, such that every fault will mask one sensitized path.

vb) Assignment of values to a type 2 subnetwork. For a type 2 subnetwork, we should apply e-vector test input patterns to all the gates whenever possible. If we can, the test will detect additional faults. We should try to make this additional detectable fault a yet undetected prime fault by carefully chosing the e-vector input pattern. This may reduce the size of the future MFTS.

vc) If the sensitized path contains a prime line k, which has a prime fault k s-a-i (i is 0 or 1), then we may have masking faults or we may detect additional faults. If

the test assigns the value i on prime line k, then the prime fault k s-a-i will be a masking fault unless the sensitized path passes through a type 2a gate after line k; in that case it will mask only one path of a reconvergent fan-out and the masking fault will be determined as in Case (IIb) of (va). If the test assigns value $\bar{i}$ on prime line k, then the test will also detect the prime fault k s-a-i, unless the sensitized path passes through a type 1a gate after line k.

vi) In step (v), we derive a test that detects a given prime fault and which determines the masking faults under the derived test. If the test detects an additional prime fault as in (vb) or (vc), then we should identify that fault's own sensitized path and find its corresponding masking faults just like in (va) and (vc), except we need not assign any values to lines since every line is assigned already. We just examine the new type 1 subnetwork and the new sensitized path as in (va) and (vc) to determine its associated masking faults.

vii) In the above steps, if we ever encounter some conflict on a certain line, then we should choose a new sensitized path and go back to step (ii); if we have exhausted all the paths we can claim the fault in question is undetectable.

As a convenient way to denote the masking faults for a given prime fault under a generated test, we can use t : f $\bar{f}_1$ $\bar{f}_2$ ... $\bar{f}_n$. That

is, the test t detects f with masking faults f_1, f_2, ..., f_n, where f_i can either be a single prime fault or a multiple prime fault.

If the test t detects f and there is no masking fault, then we say the test t detects f completely. That is to say, t will detect any fault that has f as a component of its kernel representation. On the other hand, if there are some masking faults for f under a test t, we say t detects f conditionally.

Now, say t_1 detects $f_1 \bar{f}_2$ and t_2 detects f_2 completely, then t_1 and t_2 together will also detect f_1, f_2 and their combination. Therefore, given a set of tests, we can simply sum their corresponding detectability expressions and use the Boolean algebraic simplification laws to deduce which prime faults the set of tests will detect completely. If we can find a set of tests that will detect all the prime faults completely, then the set of tests is an MFDTS. Therefore, we can use the same Boolean algebraic simplification laws to verify that a set of tests is a MFDTS once we have the complete information for each test, like we get from Procedure 2.

Theorem 3.4: In order for a set of tests to be an MFDTS, there must be at least one test in the set that detects some prime fault completely.

Proof: Assume none of the tests detects any prime fault completely. Say t_1 detects f_1 with masking fault f_2; t_2 detect f_2 with masking fault f_3, and so on. Since the number of prime faults is finite, somewhere there will be a loop, say f_1, f_2, f_3, ... , f_t, such that no test in the set can detect $f_1 f_2 f_3 \ldots f_t$. Therefore the set of tests cannot be an MFDTS.

Q.E.D.

In order for a set of tests to be an MFDTS, we need at least one test to detect one fault completely as proved by Theorem 3.4. Usually the more tests which detect prime faults completely, the more likely the set of tests will be an MFDTS. Generally speaking, we try to generate the tests for prime faults with as few masking faults as possible. Procedure 2 does this. Furthermore, Procedure 2 also attempts to detect more than one prime fault with each test in order to reduce the size of the MFDTS. Therefore, any test generated by Procedure 2 is a locally optimal test for the given prime fault under the chosen sensitized path.

In step (va), we decide which faults can be masking faults. Usually, a masking fault is composed of faults on those lines that have desensitizing values. If the masking fault is not on a prime line, we should trace back to its equivalent faults which have prime fault components only. This can always be done.

3.4 Main Algorithm for Generating a Multiple Fault Detection Test Set

With Procedures 1 and 2, we can write our main algorithm rather easily. The algorithm will produce an MFDTS for any irredundant single output combinational network. If the circuit is redundant, the algorithm will point the redundancy out to the designer.

Main Algorithm:

(1) Pick the set of prime faults by Procedure 1, call it PF. Let TS = $\emptyset$.

(2) Choose one prime fault in PF that has not been detected yet. Use Procedure 2 to generate a test that will detect the chosen prime fault together with its masking faults. If there does not exist a test to detect the given fault go to (7). If a test is found, add the test to TS.

(3) If not all the prime faults in PF been detected by our generated TS go back to (2).

(4) If TS detects all prime faults, check TS to see whether it detects all the combinations of all the prime faults. If it does, TS is an MFDTS. (See Example 3.3 for a method)

(5) If TS does not detect all the combinations of prime faults, we have a smallest subset of prime faults, called RF, such that TS does not detect a fault that contains RF. In that case, we can pick a fault f in RF and use Procedure 2 to generate another test for f, if possible. For example, we can choose a different sensitized path, etc. The new test should not have masking faults in RF. Then, add the new test to TS if the old test detects some other prime faults that the other members of TS and the new test do not detect; else replace the old test with the new one. For the new TS, go to (4) for new verification. If we cannot find a new test with the above property, continue.

(6) Pick another fault in RF and repeat step (5). If we have exhausted all the prime faults in RF and are still unsuccessful in finding a new TS such that TS will detect the RF, continue the algorithm.

(7) If the algorithm enters this step, the circuit is a redundant circuit. The redundant lines will contain those faults that cannot be detected by our generated test set. We should inform the designer that there is redundancy in the circuit.

Note: There are two heuristics which may be used in choosing a prime fault and its sensitized path in step (2) of our main algorithm. The first one is choosing the fault at the highest level (nearest to the primary input) that is still undetected by TS. By using this heuristic, the test will detect other prime faults along the sensitized path if there are any. The other heuristic is to try to use different sensitized paths for different faults, whose prime lines are inputs of the same gate. Since they use different paths, they ususally catch different prime faults that can also be detected by their respective tests. If we use the same sensitized path, since the test derived for one fault is optimal for that path, then for the other fault, we will not gain any new detectability. Though the two heuristics are not necessary, their use is strongly recommended as they will reduce the size of the generated MFDTS in most cases.

3.5 Examples

In this section, we use four examples to illustrate our algorithm. We also introduce the table method to verify that a set of tests is an MFDTS.

Example 3.3: In this simple example, we try to explain how the algebraic and table methods verify that a set of tests is an MFDTS. Also, we show how easy and efficient the algorithm is for a tree-type network.

The circuit shown is Figure 3.2 has 11 lines and five gates and therefore has $3^{11}-1$ multiple faults or 22 single faults. To check all the multiple fault exhaustively is impractical as $3^{11} \simeq 10^5$. By Schertz's collapsing equivalence, there are 12 single fault equivalence classes. By either the D-algorithm or any other standard method, we can generate an SFDTS* for this circuit. As we examine the various SFDTS, we find that some of them are also MFDTS and some are not. Naturally, since we are interested in MFDTS, we will ask how one can determine if an SFDTS is an MFDTS. Since there are $2^6 = 64$ different test patterns, there are 11250 different minimal SFDTS for that circuit. To find one which is an MFDTS, we can use either the algebraic method or the table method. Now, suppose we have

$$t_1 = 011011 \quad : \quad x_1^1$$
$$t_2 = 101000 \quad : \quad x_2^1 \overline{x_4^1}$$
$$t_3 = 110110 \quad : \quad x_6^1$$
$$t_4 = 000101 \quad : \quad x_5^1 \overline{x_3^1}$$
$$t_5 = 111111 \quad : \quad x_7^1 + x_8^1$$
$$t_6 = 100001 \quad : \quad x_3^1 \overline{x_2^1} + x_4^1 \overline{x_5^1}$$

* SFDTS = Single fault detection test set.

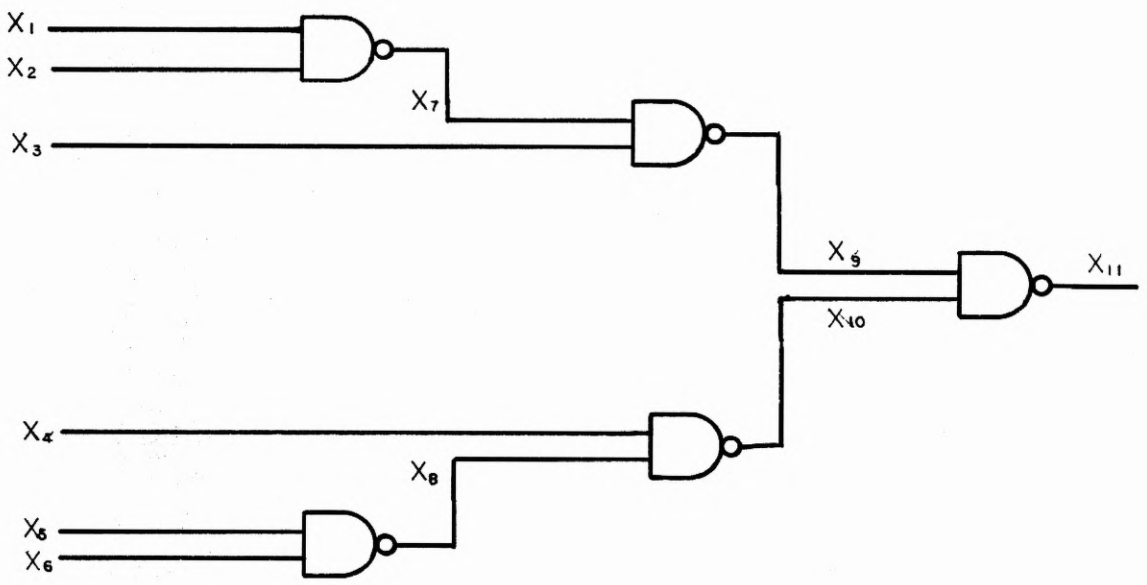


Figure 3.2

The set of tests $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ is an SFDTS. By the algebraic method, we can write

$$T: \quad x_1^1 + x_2^1 \overline{x_4^1} + x_6^1 + x_5^1 \overline{x_3^1} + x_7^1 + x_8^1 + x_3^1 \overline{x_2^1} + x_4^1 \overline{x_5^1}$$

which can not be simplified to $\Sigma_{i=1}^{8} x_i^1$. Therefore, T is not an MFDTS; for example, we can see T does not detect the fault $F = \{x_2^1, x_3^1, x_4^1, x_5^1\}$.

Now if we replace t_2 by t_2' where $t_2' = 101011 : x_2^1$, then $T' = \{t_1, t_2', t_3, t_4, t_5, t_6\}$ and

$$T' : \quad x_1^1 + x_2^1 + x_6^1 + x_5^1 \overline{x_3^1} + x_7^1 + x_8^1 + x_3^1 \overline{x_2^1} + x_4^1 \overline{x_5^1}$$

$$= x_1^1 + x_2^1 + x_3^1 + x_6^1 + x_5^1 \overline{x_3^1} + x_7^1 + x_8^1 + x_4^1 \overline{x_5^1}$$

$$= x_1^1 + x_2^1 + x_3^1 + x_5^1 + x_6^1 + x_7^1 + x_8^1 + x_4^1 \overline{x_5^1}$$

$$= x_1^1 + x_2^1 + x_3^1 + x_4^1 + x_5^1 + x_6^1 + x_7^1 + x_8^1.$$

Since T' detects all the prime faults completely, T' is an MFDTS.

A table method is shown in Table 3.1, where Table 3.1a shows why T is not an MFDTS since there is a loop consisting of $x_2^1, x_3^1, x_4^1, x_5^1$. Table 3.1b shows why T' is an MFDTS since all numerals, besides 0, can be cancelled out. We explain this method in the following paragraph.

The fault table is constructed by listing every prime fault as a column heading and every generated test as the heading of at least one row. Note, one test can head more than one row if it detects more than one prime fault. For a test t_i which detects prime fault f_j with

Prime fault	x_1^1	x_2^1	x_3^1	x_4^1	x_5^1	x_6^1	x_7^1	x_8^1
011011	0							
101000		0		1				
110110						0		
000101			1		0			
111111							0	0
100001		1	0					
				0	1			

TABLE 3.1a: Prime Fault Defection Table for SFDTS T

Prime fault	x_1^1	x_2^1	x_3^1	x_4^1	x_5^1	x_6^1	x_7^1	x_8^1
011011	0							
101011		0						
110110						0		
000101			1		0			
111111							0	0
100001		1	0					
				0	1			

TABLE 3.1b: Prime Fault Defection Table for SFDTS T'

f_k and $f_m \cdot f_n$ as its masking faults, we record a 0 in the (i,j) position to indicate that test t_i detects f_j. Different integers greater than 0 are then entered to indicate the different masking conditions. Thus, we put a 1 in the (i,k) position to indicate that fault f_k masks fault f_j under test t_i. Similarly, we put a 2 in the (i,m) and (i,n) positions to indicate that the fault $f_m \cdot f_n$ also masks fault f_j under test t_i. Therefore, all the information of detectability of the prime fault f_j under test t_i is recorded in row i. If t_i also detects another prime fault, we should use another row for that prime fault unless they have the same set of masking faults, in which case we can put them on the same row to save some space. After we record all the information about all the tests in TS, we can do something similar to reducing a covering table by crossing out certain entries in the table, as follows: We can cross out any integer besides 0 in position (i,k) if there is a 0 in position (i',k) and no other integer besides 0 appears in row i', i.e. if test $t_{i'}$ detects fault f_k completely. If the integer crossed out in position (i,k) is greater than 1, say 2, then we can also cross out any other 2 in row i. This process is repeated until no more entries can be crossed out. At this point, if there is at least one 0 in every column and no integer other than 0 appears in the table, the set of tests is an MFDTS like table 3.1b. If some non-zero integers remain in the table, a loop must exist such that if all the faults in the loop occur simultaneously they will not be detected by the set of tests under verification, as in Table 3.1a.

Now, if we apply our algorithm to Figure 3.2 we first find $x_1^1, x_2^1, x_3^1, x_4^1, x_5^1, x_6^1, x_7^1$, and x_8^1 as our prime faults. Then, we pick x_1^1 and generate the test 011011, which detects x_1^1 completely. Similarly, 101011 detects x_2^1 completely, 110101 detects x_5^1 completely, and 110110 detects x_6^1 completely. For x_3^1 we generate 000000, which detects x_3^1 and x_4^1 completely. For x_7^1, we assign $x_7 = 0$, $x_3 = 1$, $x_{10} = 1$, as $x_7 = 0 \Rightarrow x_1 = 1$ and $x_2 = 1$. Since G_5 is a type 2 gate, we would like to apply an e-test pattern to G_4. We can either apply $x_4 = 0$, $x_8 = 1$ or $x_4 = 1$, $x_8 = 0$. For $x_4 = 0$, $x_8 = 1$, we will detect x_4^1 also, but x_4^1 is detected by a previous test already. So, we choose $x_4 = 1$ and $x_8 = 0$ that also imply $x_5 = 1$, and $x_6 = 1$. Therefore, we have the test 111111 which detects x_7^1 and x_8^1 completely. We are sure this set of tests is an MFDTS, as every prime fault is detected completely.

Example 3.4: This example was first used by Poage [2] and later by Bossen and Hong [13]. The circuit is shown in Figure 3.3. By using our algorithm:

(1) We identify $x_1^1, x_2^1, x_3^1, x_4^1, x_5^1, x_6^1, x_7^1, x_8^1, x_9^1, x_{10}^1$ as our prime faults.

(2) We first pick x_1^1 and, by Procedure 2, choose lines 1,5,9, 11,13 as the sensitized path. Hence, we first assign $x_1 = 0$, $x_2 = 1$. $x_6 = 1$ and leave x_{12} unassigned. This forces $x_5 = 0$, $x_9 = 1$, $x_{11} = 1$, $x_{13} = 1$; also $x_7 = 0$ and $x_8 = 1$, $x_{10} = 0$ which imply $x_3 = 1$, $x_4 = 1$, and $x_{12} = 0$. The whole circuit has been assigned. So, we generate the test $\bar{W}XYZ$ and even though we do not have to assign any values

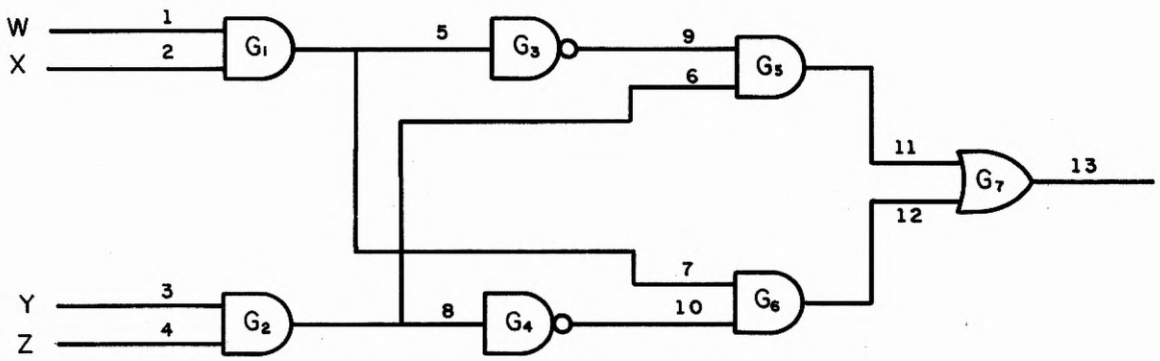


Figure 3.3

on the consistency step, we still need to go through it to find the masking faults. Although G_1 is a type 1 gate, it can not have masking faults. G_3 can not have a masking fault either, because it is a NOT gate. Only the type 1 gate G_7 can have a masking fault. G_6 does not have a $\bar{u}$-test input pattern as x_7 will be 1 due to x_1^1. Therefore, x_{10}^1 is a masking fault for x_1^1. Since x_9 is on the path of reconvergent fan-out and has a sensitizing value on it, x_9^1 is also a masking fault for x_1^1. Since x_5 is on the path of reconvergent fan-out and has a desensitizing value on it, $\bar{W}XYZ$ also detects x_5^1 which has lines 5,9,11,13 as its sensitized path. Since G_7 is type 1 gate, G_6 has a $\bar{u}$-vector input pattern, and we would like G_1 to have a $\bar{u}$-vector, too. Since this is not the case, we have a masking fault $x_1^1 \cdot x_{10}^1$ for x_5^1. Therefore, $\bar{W}XYZ$: $x_1^1 \overline{x_9^1}\ \overline{x_{10}^1} + x_5^1 \overline{(x_1^1 \cdot x_{10}^1)}$.

Now, pick x_2^1 and use lines 2,7,12,13 as the sensitized path. We assign $x_1 = 1$, $x_2 = 0$, $x_{10} = 1$, which force $x_8 = 0$, $x_6 = 0$, $x_5 = 0$, $x_9 = 1$, $x_7 = 0$, $x_{12} = 0$, $x_{11} = 0$, and $x_{13} = 0$. Since G_6 is a type 1 gate, we assign $x_3 = 0$ and $x_4 = 0$. Now, we generate the test $\bar{W}\bar{X}\bar{Y}Z$. Since G_6 is a type 1 gate on the path of reconvergent fan-out, x_8^1 and $x_3^1 \cdot x_4^1$ both will be masking faults for x_2^1. Note $x_3^1 \cdot x_4^1$ will mask x_2^1 at gate G_6. $x_3^1 \cdot x_4^1$ will change x_6 to 1 but since x_2^1 will change x_9 to 0, x_{11} will remain at 0. Therefore, $x_3^1 \cdot x_4^1$ will not remask the original fault x_2^1 at G_7.

Since x_7 has a desensitizing value on it, $\bar{W}\bar{X}\bar{Y}\bar{Z}$ will also detect x_7^1 completely since G_2 and G_4 have $\bar{u}$-vectors on them and since x_7^1 has only one path. Since G_7 is a type 2 gate and G_5 has an e-vector input pattern, $\bar{W}\bar{X}\bar{Y}\bar{Z}$ also detects x_6^1. For the fault x_6^1, G_5 is a type 1 gate, which places G_1 in a type 1 subnetwork, so, x_2^1 appears to be a masking fault. But, even if x_2^1 masks x_6^1 at gate G_5, it also remasks it at G_7 due to the fan-out. Therefore x_2^1 is not a masking fault. So, $\bar{W}\bar{X}\bar{Y}\bar{Z}$ detects $x_2^1\,\overline{x_8^1}\,\overline{x_3^1\,x_4^1} + x_6^1 + x_7^1$. Now for x_3^1, we find that $\bar{W}\bar{X}\bar{Y}\bar{Z}$ detects $x_3^1\,\overline{x_9^1}\,\overline{x_{10}^1} + x_8^1\,\overline{x_3^1}\,\overline{x_9^1}$, if we choose lines 3,8,10,12,13 as its sensitized path. If we choose lines 4,6,11,13 as the sensitized path for x_4^1, we find that $\bar{W}\bar{X}\bar{Y}\bar{Z}$ detects $x_4^1\,\overline{x_5^1}\,\overline{x_1^1\,x_2^1} + x_6^1 + x_7^1$.

For the fault x_9^1, we generate 1111 which detects $x_9^1 + x_{10}^1$.

So, the set of tests is generated, and it is shown in Table 3.2.

	x_1^1	x_2^1	x_3^1	x_4^1	x_5^1	x_6^1	x_7^1	x_8^1	x_9^1	x_{10}^1
$\bar{W}XYZ$	0								1	1
	1				0					1
$W\bar{X}\bar{Y}\bar{Z}$		0	1	1				1		
						0	0			
$W\bar{X}\bar{Y}Z$			0						1	1
			1					0	1	
$\bar{W}X\bar{Y}\bar{Z}$	1	1		0	1					
						0	0			
$WXYZ$									0	0

TABLE 3.2

This table can be reduced and all non-zero integers can be crossed out. Therefore, the test set is an MFDTS.

Example 3.5: This example is taken from Hayes [16]. The circuit is shown in Fig. 3.4. By applying the algorithm:

(1) We identify $x_1^1, x_2^1, x_3^1, x_4^1, x_5^1, x_6^1, x_7^1, x_8^1, x_9^1, x_{10}^1, x_{11}^1, x_{12}^1, x_{13}^1, x_{14}^1, x_{15}^1, x_{16}^1, x_{17}^1$, and x_{18}^1 as prime faults in the circuit.

(2) First, we pick x_1^1 and use 1,11,13,16,19 as the sensitized path. Now, we assign $x_1 = 0$, $x_2 = 1$, $x_{12} = 1$, $x_7 = 1$, $x_{15} = 1$ and leave x_{20} unassigned temporarily, since it is the input at the reconvergence point with another path. Let us use $D(\bar{D})$ to represent the value 0 (1) if the line is normal and 1 (0) if the fault occurs. So, we have $x_1 = D$, $x_{11} = \bar{D}$, $x_{13} = D$, $x_{14} = D$, $x_{16} = \bar{D}$, $x_{19} = D$. Now since x_{20} is still unassigned, assign $x_{20} = 1$. For the consistency step, since G_3 is a type 2 gate, we assign $x_3 = 0$, $x_4 = 1$. Since G_8 is a type 2 gate, we assign $x_5 = 0$, $x_6 = 1$. Since G_{10} is a type 1 gate, we would like to assign $x_{17} = 0$ and $x_{18} = 0$. But $x_{17} = 0$ forces $x_{14} = 1$ and $x_8 = 1$ which is impossible because $x_{14} = D$. Thus, we can not have $x_{17} = 0$. But, we can have $x_{17} = \bar{D}$ if we assign $x_8 = 1$. If $x_{18} = 0$, we have $x_9 = 1$ and $x_{10} = 1$. So, we have the test $t_1 = 0101011111$. Because x_{16} is on the path of reconvergent fan-out and has the sensitizing value $\bar{D}$ on it, x_{16}^1 is a masking fault

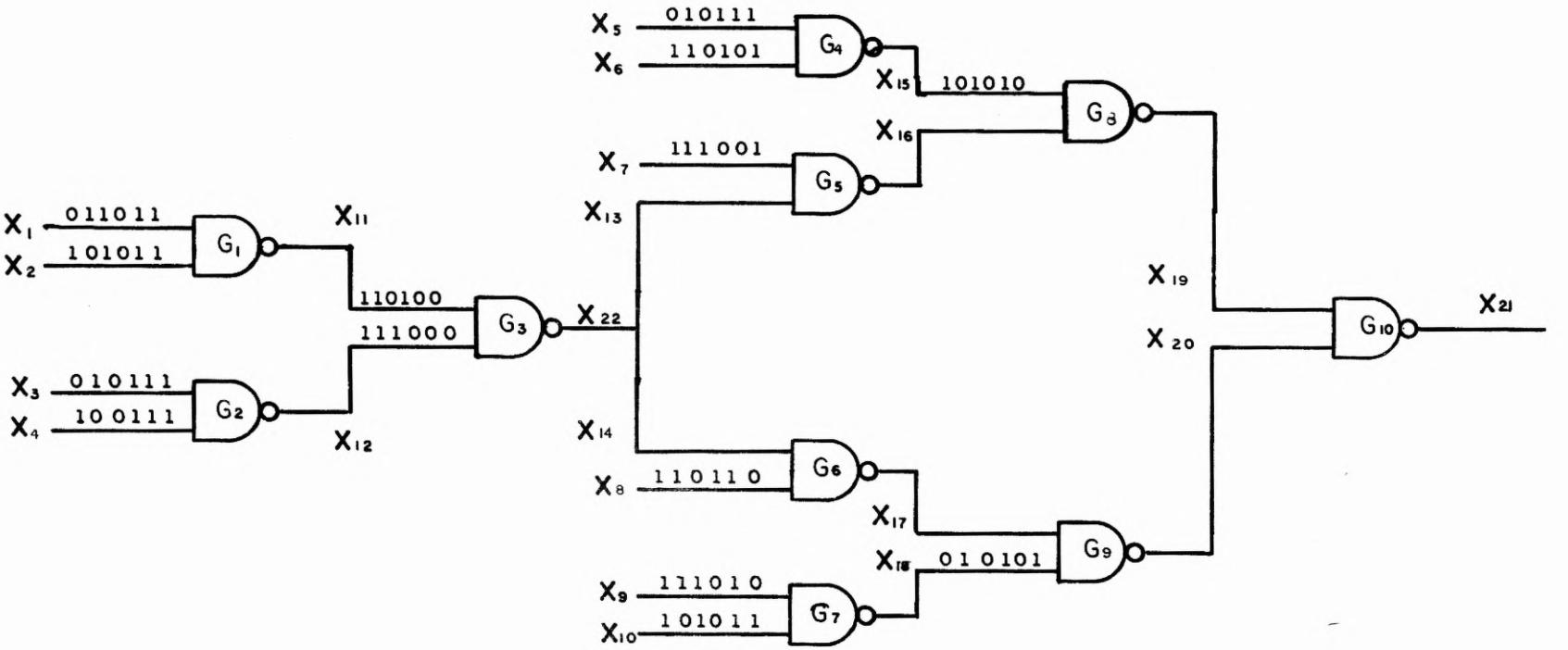

X1 011011
X2 101011
G1
X11
X3 010111
X4 100111
G2
X12
110100
111000
G3
X22
X5 010111
X6 110101
G4
X15 101010
X16
G8
X7 111001
G5
X13
X14
X8 110110
G6
X17
X18 010101
G9
X9 111010
X10 101011
G7
X19
X20
G10
X21

Figure 3.4

for x_1^1 under t_1. Since t_1 also detects x_3^1 and it has 3,12,13,16,19 as the sensitized path, x_{16}^1 is also a masking fault for x_3^1. t_1 also detects x_5^1 which uses 5,15,19 as its sensitized path and, since G_{10} is a type 1 gate, in the type 1 subnetwork with x_{20} as its output, $x_{17} = 1$ and $x_{18} = 0$ due to x_5^1. Therefore, x_{18}^1 is a masking fault for x_5^1. Since the sensitized path 1,11,13,16,19 contains the prime line x_{13} and the test requires a value 0 on x_{13}, t_1 also detects x_{13}^1, which uses x_{13}-x_{16}-x_{19} as its sensitized path. Again x_{18}^1 is a masking fault for x_{13}^1. Therefore, t_1 detects $(x_1^1+x_3^1)\overline{x_{16}^1} + (x_5^1+x_{13}^1)\overline{x_{18}^1}$.

(3) For the fault x_2^1, we heuristically choose x_2-x_{11}-x_{14}-x_{17}-x_{20} as the sensitized path and generate test t_2 = 1010111110 which detects $(x_2^1+x_4^1)\overline{x_{17}^1} + (x_{10}^1+x_{14}^1)\overline{x_{15}^1}$.

For x_{11}^1, we pick x_{11}-x_{13}-x_{16}-x_{19} as the sensitized path and generate

t_3 = 1100001011 which detects $x_{11}^1\,\overline{x_{13}^1}\,\overline{x_5^1 x_6^1} + x_{16}^1 + x_{18}^1\,\overline{x_8^1}$

For x_{12}^1, we heuristically choose x_{12}-x_{14}-x_{17}-x_{20} as the sensitized path, and generate

t_4 = 0011110100 which detects $x_{12}^1\,\overline{x_{14}^1}\,\overline{x_9^1 x_{10}^1} + x_{17}^1 + x_{15}^1\,\overline{x_7^1}$.

For x_6^1, x_6-x_{15}-x_{19} is the only path, and we generate

t_5 = 1111100111 which detects $x_6^1 + x_7^1$.

For x_8^1, x_8-x_{17}-x_{20} is the only path, and we generate

t_6 = 1111111001 which detects $x_8^1 + x_9^1$.

(4) The tests are listed in Table 3.3. From the table, we can see $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ is an MFDTS.

For this example, we easily generated the MFDTS for the circuit. Actually, we get an optimal solution. If we apply the algorithm of Bossen and Hong [13] to the circuit, we will get either 13 tests or 22 tests in its MFDTS depending on whether an "abnormal true" test or an "abnormal false" test is used. Our algorithm yields an MFDTS which requires only 6 tests and, therefore, a much better solution.

Example 3.6: The circuit shown in Fig. 3.5 taken from Danderpani et al. [20] is single-line irredundant, but four-line redundant. We use this example to illustrate how our algorithm can pin-point redundancy. From our algorithm:

(1) We get PF = $\{x_i^0 | i=1,2,\ldots,18$ and $x_j^1 | j=19,20,\ldots,33\}$.

(2) Using Procedure 2, we generate all the tests together with their masking faults. The result is shown in Table 3.4.

(3) From Table 3.4, we have the smallest loop RF = $\{x_1^0, x_5^0, x_{21}^1, x_{24}^1\}$, such that all the prime faults in RF mask each other. So the test set is not an MFDTS.

(4) Now, we pick $x_1^0 \in$ RF, and we find that test 1011101 is the only test for x_1^0. Next we pick $x_5^0 \in$ RF and find any test that detects x_5^0 will have a masking fault x_{21}^1. Similarly, for x_{21}^1 and x_{24}^1, we cannot find any new test that will have masking faults other than the faults in RF.

	x_1^1	x_2^1	x_3^1	x_4^1	x_5^1	x_6^1	x_7^1	x_8^1	x_9^1	x_{10}^1	x_{11}^1	x_{12}^1	x_{13}^1	x_{14}^1	x_{15}^1	x_{16}^1	x_{17}^1	x_{18}^1
0101011111	0		0													1		
					0								0					1
1010111110		0		0													1	
										0				0	1			
1100001011					1	1					0		1					
																0		
								1										0
0011110100									1	1		0		1				
																	0	
							1								0			
1111100111						0	0											
1111111001								0	0									

TABLE 3.3

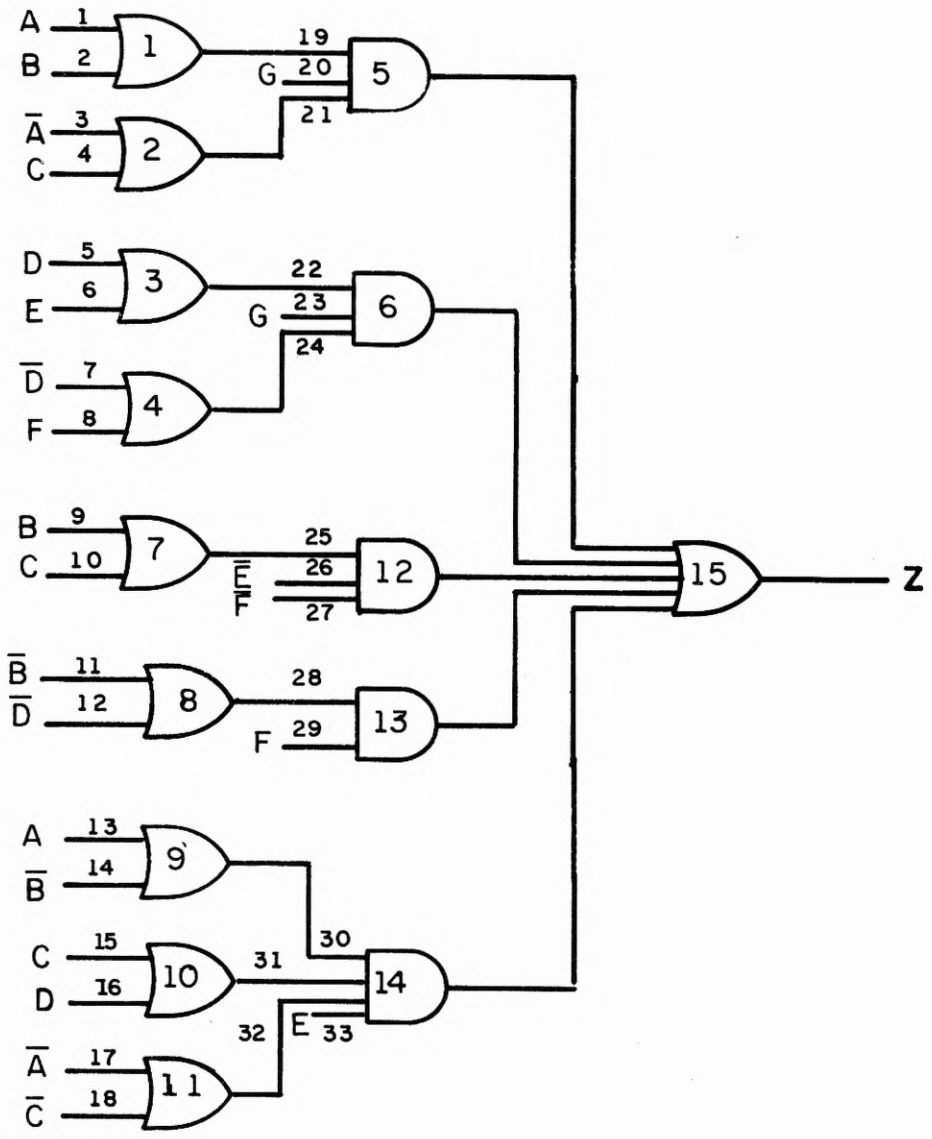


Figure 3.5 A Redundant Circuit

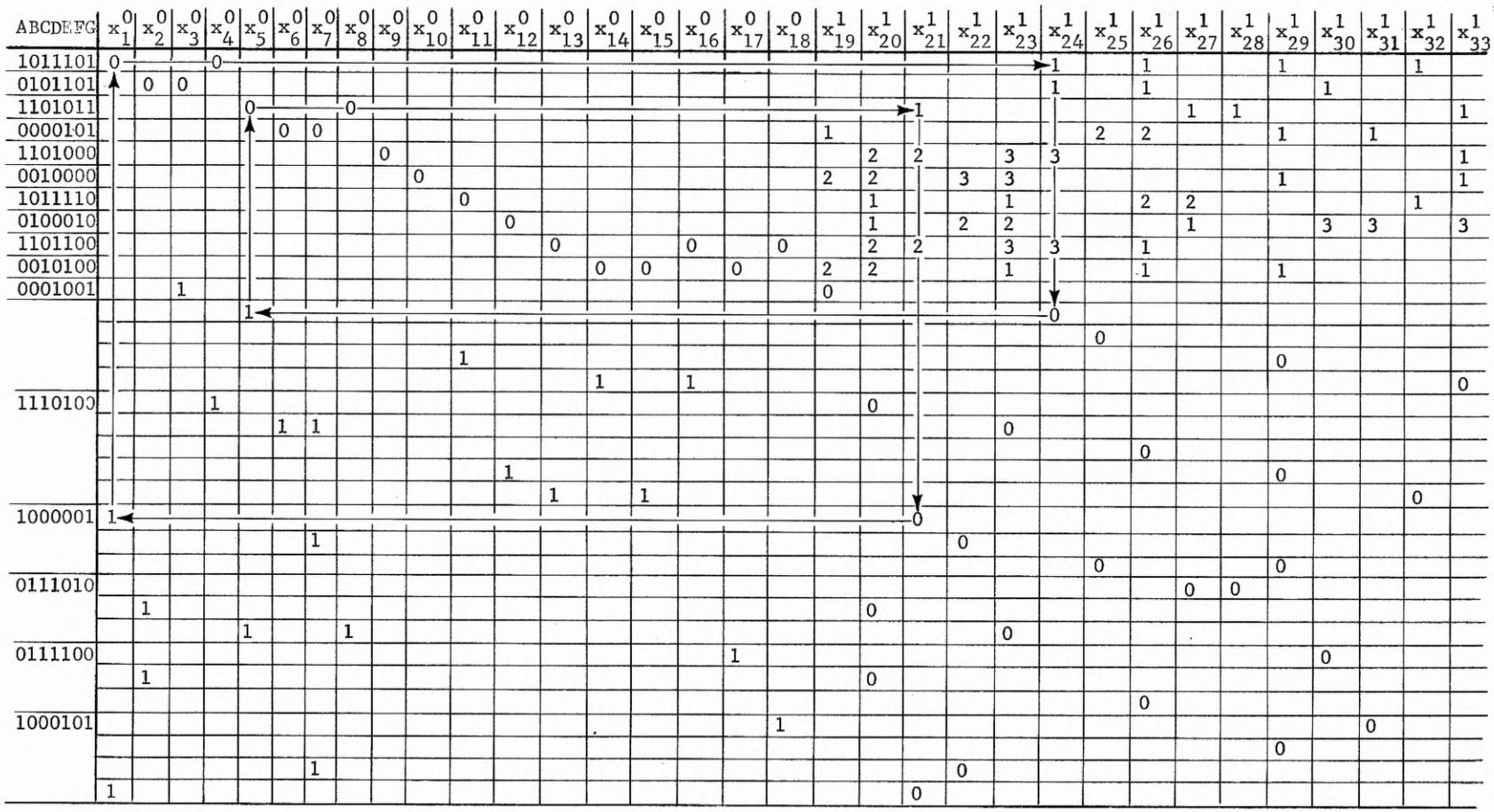

ABCDEFG
x_1^0 x_2^0 x_3^0 x_4^0 x_5^0 x_6^0 x_7^0 x_8^0 x_9^0 x_{10}^0 x_{11}^0 x_{12}^0 x_{13}^0 x_{14}^0 x_{15}^0 x_{16}^0 x_{17}^0 x_{18}^0 x_{19}^1 x_{20}^1 x_{21}^1 x_{22}^1 x_{23}^1 x_{24}^1 x_{25}^1 x_{26}^1 x_{27}^1 x_{28}^1 x_{29}^1 x_{30}^1 x_{31}^1 x_{32}^1 x_{33}^1
1011101
0101101
1101011
0000101
1101000
0010000
1011110
0100010
1101100
0010100
0001001
1110100
1000001
0111010
0111100
1000101

TABLE 3.4

Therefore, the loop RF can not be broken.

(5) The circuit is a multiple-line redundant circuit; the redundancy consist of lines 1,5,21,24. These lines can be removed, together with any gates that as a consequence become detached from the circuit (e.g. gate 2) or equivalent to a straight-through connection (e.g. gate 1). The reduced circuit is shown in Fig. 3.6 and its MFDTS in Table 3.5.

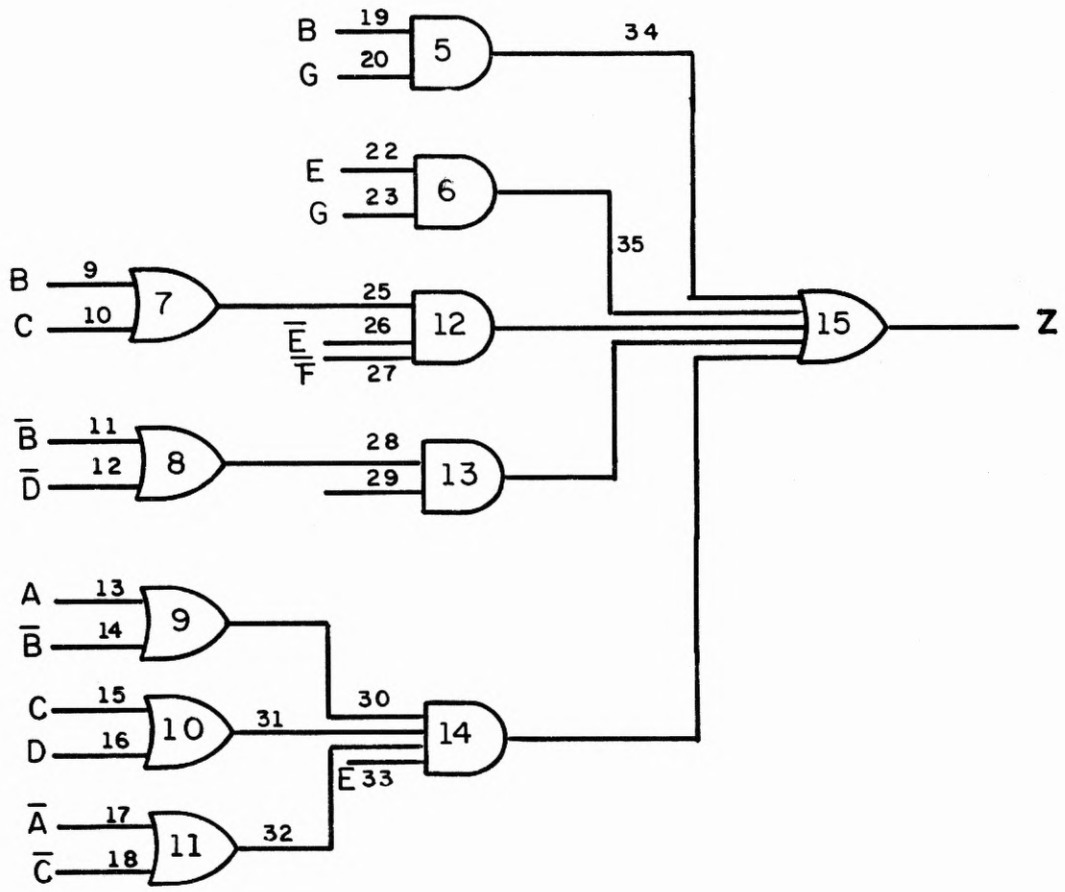


Figure 3.6 The Reduced Circuit of Figure 3.5

ABCDEFG	x_9^0	x_{10}^0	x_{11}^0	x_{12}^0	x_{13}^0	x_{14}^0	x_{15}^0	x_{16}^0	x_{17}^0	x_{18}^0	x_{19}^1	x_{20}^1	x_{22}^1	x_{23}^1	x_{25}^1	x_{26}^1	x_{27}^1	x_{28}^1	x_{29}^1	x_{30}^1	x_{31}^1	x_{32}^1	x_{33}^1	x_{34}^1	x_{35}^1
0101000	0											1								1			1		
1010000		0																	1			2	1		
0001010			0												1		1						1		
0100110				0								1		1		1	1			3	3				
1101110					0			0		0		1		1		1	1	1							
0010100						0	0		0					1	0	1			1						
0000001											0	0	0		0				0						
1100100	1			1	1					1						0					0				
1101010	1				1			1		1		0					0						0		
0111100									1			0		0		0				0					
1111100					1									0		0						0			
0101011													1				1	1		1			2	0	
1010101											1					1			1			1			0

TABLE 3.5

CHAPTER 4

SOME PROPERTIES AND GENERALIZATIONS OF THE ALGORITHM

4.1 Introduction

In this chapter, an important theorem about multiple-line redundancy is presented. This serves as the basis for an earlier claim made about the algorithm. Some related and interesting theorems are given new proofs via the fault masking technique. The algorithm is generalized to multiple output combinational networks. Fault location is also discussed. Finally, some comparisons between the algorithm and other existing algorithms are made.

4.2 A Central Theorem

Single-line redundancy of a network was defined in Chapter 2. As our algorithm in Chapter 3 is concerned with multiple faults, a more precise definition for a multiple-line redundancy of a network is given as follows.

Definition 4.1: A network is called n-line redundant ($n>1$), if there is a set of n independent lines in N whose removal does not affect the output function, but removal of a set of n-1 or fewer independent lines from the original circuit will affect the output function.

Equivalently, a network is called n-line redundant ($n>1$) if it has a multiple fault with n components that is undetectable, but any fault with n-1 or fewer components is detectable.

In Chapter 3, step 5 of the main algorithm identifies a subset RF of all the prime faults such that the fault or faults in RF mask each other under any test. Then, we claim the multiple fault with all the components of RF is not detectable and hence, by the above definition, the circuit is n-line redundant if the smallest cardinality of RF is n. In generating the test for each prime fault, in using the steps in the algorithms, we merely exhaust all the possible tests for detecting a single prime fault. Therefore, for the set RF, we only consider all the possible tests that can detect a single component of RF. One may ask the question: Is it possible that there exists a multiple fault such that among all the tests that detect it none of them will detect any single component of the fault? In other words, if we exhaust all the tests that detect every single component of the multiple fault and none of them will detect the multiple fault, is it possible that the multiple fault is not redundant?

The answer to the above question is no. This is proved by the following theorem.

Theorem 4.1: In a single-output combinational network, let $F = \{f_1, f_2, \ldots, f_n\}$ be any multiple fault. Let T be the complete set of tests such that every member in T detects F. Let T_i be the complete set of tests such that every member in T_i detects f_i, $i = 1,2,\ldots,n$. Assume f_i, $i = 1,2,\ldots,n$, and F are irredundant, i.e. $T_i \neq \emptyset$, and $T \neq \emptyset$. Then there is at least one test $t \in T$ such that $t \in T_i$ for some i $1 \leq i \leq n$.

Proof: Without loss of generality, we can assume the network is a normal NAND network, the multiple fault F is in its canonical form, and F has at least two components. Also, assume no test in T is also in T_i for some $1 \leq i \leq n$.

Let $t \in T$; t will partition the fault F into two sets F_1 and F_2 such that $F_2 = \{f \mid f \in F$ and the occurrence of f will not affect the detection of F by t$\}$ i.e. t also detects $F-F_2$, and $F_1 = F-F_2$. Therefore, t detects F_1. Since t detects F, $F_1 \neq \emptyset$.

Let $f_i \in F_1$ and ℓ_i be the line associated with it. By assumption, t does not detect f_i. Along every path from f_i to the primary output there exists a gate G_i such that G_i has at least two inputs with value 0 on them; call them ℓ_{i1} and ℓ_{i2} where ℓ_{i1} is the line that is part of the path from f_i to G_i, and ℓ_{i2} is the line that is part of the path from some fault $f_j \in F_1$ to G_i. (If it is not, $f_i \in F_2$). Now, if we can change the value of ℓ_{i2} from 0 to 1 by changing all the values along the path from f_j to G_i without affecting the path sensitization from f_i to G_i, then the new test t' will also detect F, except t' will partition F in a way such that $f_j \in F_2$ since ℓ_j has the value 1 for t'.

For t', we can still choose $f_i \in F_1$ and repeat the above argument. Because we assume no test in T will detect f_i, there must exist a case such that we cannot change the value on ℓ_{i2}. In other words, in order to sensitize f_i from ℓ_i to G_i, the line ℓ_{i2} must be 0. This will desensitize the gate G_i. Because the sensitized path pattern is unique along any one path, the fault f_i cannot be detected along that path.

But, our argument can apply to all the paths from ℓ_i to the primary output. Hence, no test will be able to detect f_i by our assumption. This contradicts our assumption that f_i is irredundant. Therefore, our assumption that no test in T will detect any single fault f_i $1 \leq i \leq n$ of F is wrong. This completes our proof.

Q.E.D.

Note 1: The above theorem is true only for some fault component of F, namely only for those faults in F_1. It is not true for every component of F. We can see this by the following example.

Example 4.1: The circuit in Fig. 4.1 is an irredundant circuit. Consider the fault F = {a/1,d/1}.

The set T = {(0000),(0100)} is the set of tests that detects F. But none of the tests in T detects a/1. Of course, both of them also detect d/1, as both tests in T partition F into F_1 and F_2 where F_1 = {d/1} and F_2 = {a/1}.

Note 2: For a single line redundant circuit, the above theorem is also not true as we can see from Example 4.2.

Example 4.2: The circuit in Fig. 4.2 is a redundant circuit. Lines 2 and 7 are both redundant lines. Obviously no test will detect either 2/1 or 7/1. But the multiple fault F = {2/1,7/1} is detected by test 1001.

With Theorem 4.1, we can prove the following theorem.

Theorem 4.2: A multiple fault F is undetectable in a single output combinational network if and only if for every subfault F_1 of F, there is another subfault F_2 of F such that F_2 will mask F_1 under the test that

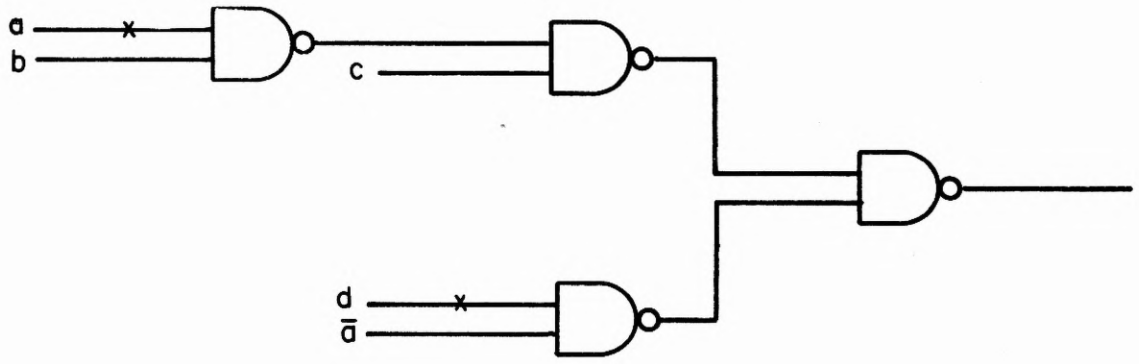


Figure 4.1

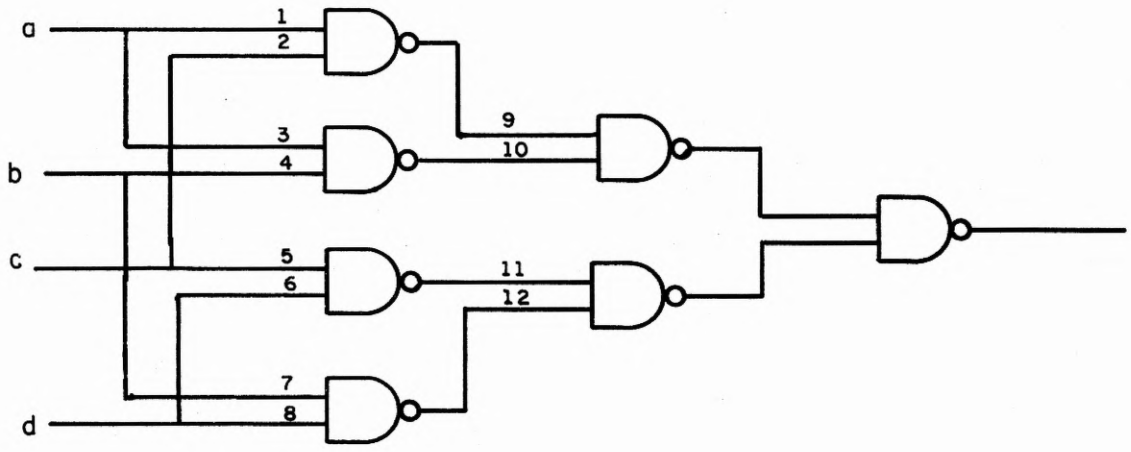


Figure 4.2

detects F_1.

Proof: Suppose the multiple fault F is undetectable in a single output combinational network, then for any subfault F_1 of F if F_1 is not redundant, i.e. there is a test t that detects F_1, then there must exist a subfault F_2 of F such that F_2 will mask F_1 under test t, because t cannot detect F.

Conversely, for any subfault F_1 of F, there is a subfault F_2 of F such that F_2 will mask F_1 under the test that detects F_1. Now, assume F is detectable, i.e. there is a set of tests T that detects F. By Theorem 4.1, there must exist a test t in T such that t also detects some $f \in F$. Let $F_1 = \{f\}$; by our condition, there is a subfault F_2 of F such that F_2 will mask F under t. Since t does not detect $F_1 \cdot F_2$, and $F_1 \cdot F_2$ is a subset of F, t will not detect F, either. This contradicts $t \in T$. Therefore, F is undetectable.

Q.E.D.

We know a set of test is an MFTS if it detects all the prime faults and their combinations. By this concept, we can give an easy proof for the following theorem.

Theorem 4.3: For a restricted fan-out-free circuit with primary fan-out allowed, every SFTS is also an MFTS.

Proof: A generalized restricted fan-out-free circuit with primary fan-out allowed is shown in Fig. 4.3. Assume the circuit has p+1 levels. Let G be the gate in level 0 and let $G_{p1}, G_{p2}, \ldots, G_{pn_p}$ be the gates in level p.

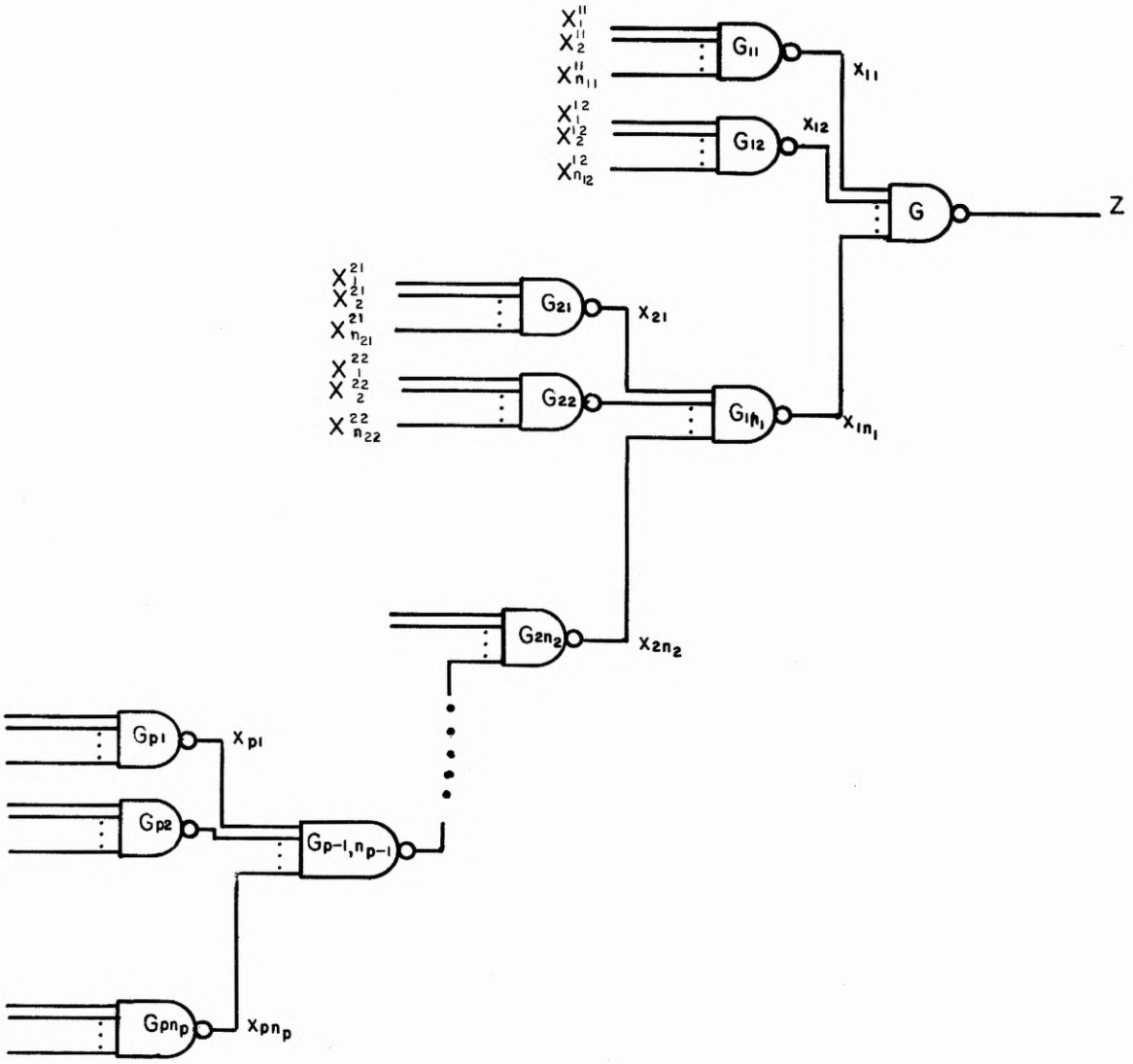


Figure 4.3 Restricted Fan-out Free Network

From the circuit, we can see all the lines have a prime fault associated with them except the primary output line and the lines x_{in_i} $i = 1,2,\ldots,p-1$.

Now, we are going to prove any SFTS of the circuit in Fig. 4.3 is also an MFTS. Since the SFTS will detect all the single faults, it obviously detects all the prime faults. If we can prove it also detects all the combinations of the prime faults, then it will be an MFTS.

Let us look in detail at those tests that detect certain prime faults. First of all, consider the inputs of gate G_{1j}, $j=1,\ldots,n_1-1$. Any test that detects $x_1^{11}/1$ will apply 0 on x_1^{11} and 1 on all x_j^{11}, $j=2,\ldots,n_{11}$. Also it will apply 1 to x_{1j}, $j=2,\ldots,n_1$. Hence $G_{\cdot}$ is a type 2 gate and no masking fault can exist. Therefore any test that detects $x_1^{11}/1$ will detect it unconditionally. Similarly, any test that detects $x_i^{1j}/1$ where $j=1,2,\ldots,n_1-1$, $i=1,2,\ldots,n_{1j}$ will detect it unconditionally.

Now for level 2 primary inputs, say x_1^{21}, any test that detects $x_1^{21}/1$ will apply e-test patterns to gates G_{21} and G, and a u-test pattern to gate G_{1n_1}. Therefore masking faults can occur only at the co-inputs of x_1^{21} and x_{1n_1}. As the co-inputs of x_1^{21} are 1's already, masking faults can exist only on lines x_i^{1j} where $j=1,2,\ldots,n_1-1$, $i=1,2,\ldots,n_{1j}$. We have tests in the SFTS that detect x_i^{1j} unconditionally for all j and i. Therefore, the SFTS will also detect $x_1^{21}/1$ unconditionally. Similarly the SFTS will detect $x_i^{2j}/1$, $j=1,2,\ldots,n_2-1$, $i=1,2,\ldots,n_{2j}$ unconditionally.

For level 3 up to level p, any test that detects a primary input line fault at level q can only have masking faults on level q-1 or lower.

Inductively, the SFTS detects any fault from level 1 to q-1 completely. Therefore, the SFTS will detect that fault at level q unconditionally. So far, we have proved the SFTS will detect all the prime faults on primary input lines unconditionally. We still have to check the prime faults that are not on primary input lines, namely those on line x_{ij} where $i=1,2,\ldots,p-1$, $j=1,2,\ldots,n_i-1$, and $i=p$, $j=1,2,\ldots,n_p$, or those on the output lines of all the prime gates.

For $x_{pj}/1$, $1\leq j\leq n_p$, we must use a test that applies 0 on x_{pj} and 1 on x_{pi}, $i\neq j$. Also it must apply 1 on all the output lines of all the prime gates.

Only those faults that are either on input lines of gate G_{pi}, $i\neq j$, or on other primary input lines of lower levels could become masking faults for $x_{pj}/1$ under any test that detects $x_{pj}/1$. But, as we proved before, the SFTS will detect all faults on all the primary input lines unconditionally. Therefore the SFTS will detect $x_{pj}/1$ unconditionally for all $j=1,2,\ldots,n_p$.

Now, consider prime gates on level p-1. For any prime fault $x_{p-1j}/1$, $j=1,2,\ldots,n_{p-1}-1$, only those faults that are either on primary input lines of level p-1 or lower, or faults $x_{pj}/1$ could become masking faults under any test. But, we just proved SFTS detects $x_{pj}/1$ unconditionally. So, SFTS will detect $x_{p-1j}/1$ unconditionally, too, for any $j=1,2,\ldots,n_{p-1}-1$.

The above argument can be repeated from level p-1 to p-2 until level 1. We can say SFTS detects $x_{ij}/1$ unconditionally for $i=1,2,\ldots,p-1$, $j=1,2,\ldots,n_i-1$.

Now, we have included all the prime faults and we have proved any SFTS detects all the prime fault unconditionally. Therefore the SFTS is also an MFTS.

Q.E.D.

As we know, redundancy is very closely related to the detectability of faults. From the above theorem, we can easily prove the following corollary.

Corollary: If a restricted fan-out-free network with primary input fan-out allowed is a single-line irredundant circuit, then it is an irredundant circuit.

Proof: In a single-line irredundant circuit, every single fault is detectable and so the circuit must have at least one SFTS. By Theorem 4.3, we know the SFTS is also an MFTS for that circuit. Hence every multiple fault in that circuit is detectable and the circuit must be irredundant.

Q.E.D.

4.3 Application of the Algorithm

The algorithm we developed in Chapter 3 is an efficient tool for generating an MFTS for an irredundant single output combinational network. In this section, we will discuss some applications of the algorithm besides generating an MFTS.

First of all, the algorithm can pin-point any redundancy in the network. This includes multiple-line redundancy as well as single-

line redundancy. If a fault can not be detected by the algorithm we use, the set of lines associated with that fault must be redundant. For multiple-line redundancy detection, as in Example 3.6, this is believed to be the only existing practical method.

In fault tolerant computing redundancy is sometime added intentionally, as for example in Triple Modular Redundant (TMR) systems to increase the reliability of the whole system during certain short time intervals. However, random and unwanted redundancy created by a logic designer will not only increase the manufacturing cost, but will add difficulty to fault detection. It is highly recommended for a logic designer to eliminate unwanted, hidden redundancy. The algorithm clearly can be used to accomplish this.

Another application of the algorithm can be described as follows. For a given network and a given set S of diagnostic tests, we might want to determine if this set is an MFTS. If it is not, we might like to know which faults S does not detect. We can answer these questions by applying the algorithm, and for every test, determining which prime faults are detected together with their corresponding masking faults. Note - in this case, we omit the generation test step as the test is given. After all the tests in S have been analyzed, we can use either the table method or the algebraic method described in Chapter 3 to determine if S is an MFTS. Example 3.3 in Chapter 3 illustrates this application.

Also, in certain cases, we may want to know whether a set of tests will detect a certain set of faults in a network. The algorithm can be easily applied to answer this question with some minor modifications.

4.4 Generalization to Multiple Output Combinational Circuit

In generating an MFTS for a multiple output combinational network, the idea and philosophy behind the algorithm are the same, except we should make some modifications in some of our definitions.

Definition 4.2: A fault in a multiple output network is said to be detected if it is detected via at least one of the outputs.

Definition 4.3: A set of lines is redundant if the fault associated with the lines cannot be detected via any output.

We will still use the notation Z_N as the output function of N. For multiple output circuits, Z_N is a vector instead of a scalar.

One important difference between single output and multiple output circuits is that there are non-reconvergent fan-outs in multiple output circuits. Whenever we are dealing with multiple output circuits, we should distinguish the two different types of fan-out. We can treat non-reconvergent fan-out just as ordinary lines in the cirucit. For example, any branch line of a non-reconvergent fan-out will not be chosen as prime fault line unless the fan-out comes from the primary input line or the output of a prime gate. Other procedures in the algorithm will be the same except we should always try to sensitize one path for

every output line. Using this heuristic, the size of the MFTS should be decreased. An example is given below.

Example 4.3: In the network with primary outputs z_1 and z_2 shown in Fig. 4.4, we have a non-reconvergent fan-out, in which branch lines 8 and 9 go to outputs z_1 and z_2, respectively.

Therefore, for this network, we will choose lines 1,2,3,4 5,6,7,10,11,12 as the lines that have prime faults on them.

By a procedure similar to our algorithm in Chapter 3, we get the result shown in Table 4.1.

From Table 4.1, we can see we have an MFTS = {(010001), (101001), (110011), (011101), (111110)}. Therefore, the algorithm in Chapter 3 can be easily generalized to a multiple-output combinational network.

XYZUVW	1/1	2/1	3/1	4/1	5/1	6/1	7/1	10/1	11/1	12/1
010001	0			0						
101001		0	0							
110011				1	0					
								0		
011101	1					0				
									0	
111110							0			0

TABLE 4.1

4.5 Fault Location

Fault location in a network consist of generating a set of tests that, when applied to the network, identifies which fault has

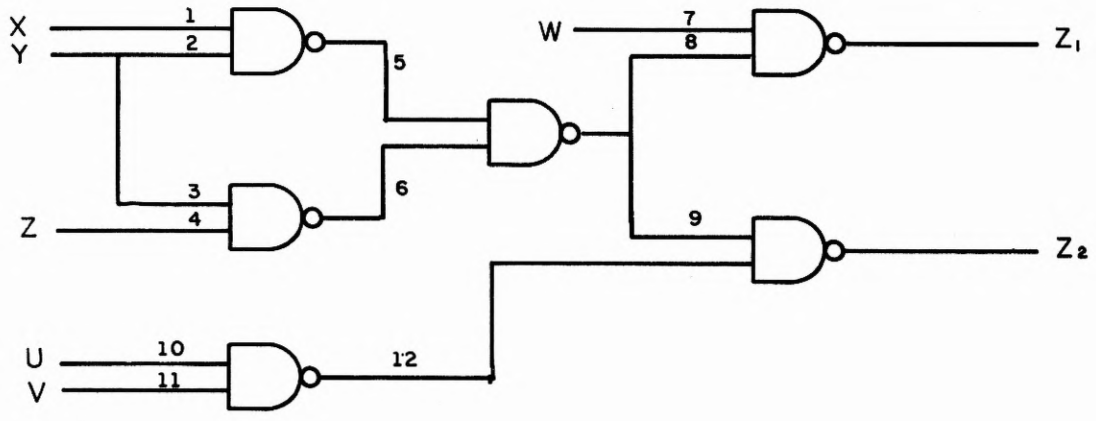


Figure 4.4 A Multiple Output Combinational Network

occurred in the network. Unfortunately, due to fault equivalence, all faults within an equivalence class can not be distinguished by any tests. So, the best we can do in fault location is to distinguish all the fault equivalence classes.

The determination of all the single-fault equivalence class has been discussed in detail in Chapter 2. If we can find a set of tests such that every test will detect just one single fault, then the set of tests will be a location test set for the single-fault equivalence class.

To find all the multiple-fault equivalence classes would be a very difficult and tedious job. For a network with n primary input lines, the number of multiple fault equivalence classes is in the vicinity of 2^n. Since there are altogether 3^m-1 multiple faults for a network with m lines and m is usually far greater than n, the average size of an equivalence class is very large. For the very simple network shown in Fig. 4.5, there is an average of 136 faults in every equivalence class.

For a fan-out-free network, the number of multiple fault equivalence classes is exactly 2^n if the network has n inputs. An example is shown in Fig. 4.5, and the equivalence classes are listed in Table 4.2.

Because the number of equivalence classes increases exponentially with the number of inputs, one may tend to think it is very difficult to find a complete fault location test. This is not true as we shall

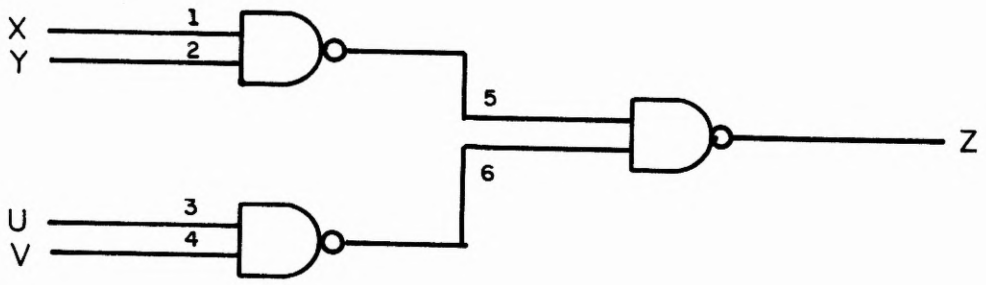


Figure 4.5

	FAULT REPRESENTATIVE OF MULTIPLE FAULT CLASS	CORRESPONDING FAULT FUNCTION
1	1/1	Y + UV
2	2/1	X + UV
3	3/1	XY + V
4	4/1	XY + U
5	5/1	UV
6	6/1	XY
7	(1/1 and 2/1) or (3/1 and 4/1)	1
8	1/1 and 3/1	Y + V
9	1/1 and 4/1	Y + U
10	1/1 and 6/1	Y
11	2/1 and 3/1	X + V
12	2/1 and 4/1	X + U
13	2/1 and 6/1	X
14	3/1 and 5/1	V
15	4/1 and 6/1	U
16	5/1 and 6/1	0

TABLE 4.2

see from some simple calcaultions. For a network with n inputs, there are about 2^n fault equivalence classes. But, there are also 2^n input tests. The number of combinations of responses to all the tests can be theoretically as high as 2^{2^n}. With all these combinations, we should be able to distinguish all the fault equivalences. One obvious question arises: how should we choose a minimum set of tests, say m tests, such that $2^m \geq 2^n$ and every fault class will have a different response? We want a minimum set of tests because we can not spend the computer time required for all the tests.

Our algorithm in Chapter 3 provides a good way of finding a complete fault location test set. If a test detects more than one fault, say it detects f_1 and f_2, then a response to this test which is different from the response for a fault-free circuit can not tell us exactly which fault has occurred or whether both have occurred if they do not mask each other. Therefore, from a fault location point of view, we should use a test that will detect only one prime fault, and, hopefully, the test will detect it unconditionally. Since any multiple fault can be represented as the combination of one or more prime faults, all the different multiple fault classes can be represented by either a prime fault or a combination of prime faults. If we can find a set of tests such that every test will detect only one prime fault unconditionally, then we have a minimum fault location test set.

For a fan-out-free network, we can always find a test that will detect only one prime fault unconditionally. Therefore, we can easily find the location test set for a fan-out-free network. A location test set for Fig. 4.5 is listed in Table 4.3, and its response under different fault classes is listed in Table 4.4.

It is very important to generate a test in the location test set that detects only one prime fault, if possible. Therefore, we should always assign the optimally desensitizing vectors, namely the $\bar{u}$ input patterns, to all the gates, even type 2 gates, whenever possible. If a test t which is generated to detect fault f_1 will also detect another prime fault f_2, then we still want to include another test that

	XYUV	1/1	2/1	3/1	4/1	5/1	6/1
t_1	0100	0					
t_2	1000		0				
t_3	0001			0			
t_4	0010				0		
t_5	1100					0	
t_6	0011						0

TABLE 4.3: A COMPLETE FAULT LOCATION TEST

FAULT CLASS	t_1	t_2	t_3	t_4	t_5	t_6 TEST RESPONSE
Fault free ∅	0	0	0	0	1	1
1	1	0	0	0	1	1
2	0	1	0	0	1	1
3	0	0	1	0	1	1
4	0	0	0	1	1	1
5	0	0	0	0	0	1
6	0	0	0	0	1	0
7	1	1	1	1	1	1
8	1	0	1	0	1	1
9	1	0	0	1	1	1
10	1	0	0	0	1	0
11	0	1	1	0	1	1
12	0	1	0	1	1	1
13	0	1	0	0	1	0
14	0	0	1	0	0	1
15	0	0	0	1	0	1
16	0	0	0	0	0	0

TABLE 4.4: THE RESPONSE OF THE COMPLETE FAULT LOCATION TEST FOR FIGURE 4.5

can detect f_2 alone, if such a test exists. From the above discussion, we can see that generating a complete location test set is quite straightforward, and the number of tests is about equal to the number of prime faults.

If in the application of a multiple fault location test we get a response other than one of those corresponding to one of the fault classes, say 010101 for Fig. 4.5 with the test set in Table 4.3, then we can conclude that the fault in the circuit is not represented by the stuck-at fault model. We should use another method to determine the fault.

Once a fault is pin-pointed to within a fault class, there is nothing more that can be done to tell exactly which fault in the class has occurred if normal tests are used. For higher diagnostic resolution, other technologies or methods like test point insertion, line isolation, etc, would have to be used.

In modern integrated circuit technology, one is usually more interested in determining which chip or board is faulty instead of which line or lines are stuck. With this idea in mind, we should apply tests that will detect faults in one chip only. In this way, we can distinguish faults on different chips. But, within a chip, the test should detect as many prime faults as possible. Thus, we can minimize the number of tests if the combinations of faults detected by a given test from their classes within a single chip.

Once a circuit is divided into different modules or blocks, one can easily find the location test set for the blocks with a small modification to the algorithm.

4.6 Comparison Between this Algorithm and Others

Since this algorithm is completely different from other existing algorithms for generating the MFTS for a combinational network, it is not very easy to compare this algorithm with others. But, it can be shown that this algorithm satisfies all the general criteria that others fail. An algorithm must

a) Generate a set of tests that will detect all specified faults;

b) Generate a test set efficiently;

c) Generate as small a test set as possible;

d) Be able to generate test sets for all circuits.

The algorithm described in Chapter 3 satisfies all the above criteria. Let us take a look at other methods.

The first theoretical method for generating a complete multiple fault detect test set for a combinational method is described by Poage and McCluskey [2]. They first find an SFTS for the network and then find so called "critical columns", i.e. those multiple faults that are not detected by the SFTS, and then generate additional tests to form an MFTS. This method requires a huge amount of calculation and hence

violates Criterion b). Therefore, the method is not considered to be practical for actual use.

Yau and Tang [12] use Boolean differences to generate an MFTS. Their method is based on the concept that if a test for a single path passes, then there is no fault on that path. So, if a test set verifies all the possible paths from every primary input line to every primary output line, then it is an MFTS. Even if we ignore the complexity of Boolean differences in computation, their method can not be applied if the circuit either has at least two paths from every primary input terminal to the primary output terminal, as in Example 3.6, or has paths which need to be blocked according to the scheme, but which can not be successfully blocked. In these two cases, Yau and Tang have to turn to Poage's method, which is well-known to be very complicated and unpractical for actual use, to find tests for certain specific multiple faults. Also, the size of the MFTS that Yau and Tang generate is far larger then necessary.

Bossen and Hong [13] use Cause-Effect analysis for MFTS generation. They use a simplified cause-effect equation to find an abnormal true test set and an abnormal false test set to form an MFTS. Their method is a simplification of Poage's method. For irredundnat networks, they define checkpoints similar to our prime faults, then from either the abnormal true test set or the abnormal false test set they find an exclusive normal input cause test set to cover all the check points. The union of the two sets forms an MFTS. Some drawbacks of their algorithm are:

1) The cause-effect equations are cumbersome.

2) If the cause-effect equation is long, as it usually is, there is a long list of x_i terms; this will produce more x_j and $\hat{x}_j$ terms. It will then be tedious to find a minimum cover for a long $\hat{x}_j$ list.

3) The number of tests it generates is still very much larger than necessary, as we can see by the example below.

4) The simplified method can not be applied to redundant networks.

5) The tests it generates do not give any information as to which faults they detect. Thus, an extension of the method to fault location would be difficult.

Example 4.4: Let us apply the Bossen and Hong method to the circuit shown in Example 3.5. We can find an MFTS of six tests easily, as shown previously.

From their method, we will get:

$$F^1 = \bar{x}_5 \bar{x}_7 {}^5_n {}^7_n + \bar{x}_1 \bar{x}_3 \bar{x}_5 {}^1_n {}^3_n {}^5_n {}^{13}_n + \bar{x}_1 \bar{x}_4 \bar{x}_5 {}^1_n {}^4_n {}^5_n {}^{13}_n + \bar{x}_2 \bar{x}_3 \bar{x}_5 {}^2_n {}^3_n {}^5_n {}^{13}_n$$
$$+ \bar{x}_2 \bar{x}_4 \bar{x}_5 {}^2_n {}^4_n {}^5_n {}^{13}_n + \bar{x}_6 \bar{x}_7 {}^6_n {}^7_n + \bar{x}_1 \bar{x}_3 \bar{x}_6 {}^1_n {}^3_n {}^6_n {}^{13}_n + \bar{x}_1 \bar{x}_4 \bar{x}_6 {}^1_n {}^4_n {}^6_n {}^{13}_n$$
$$+ \bar{x}_2 \bar{x}_3 \bar{x}_6 {}^2_n {}^3_n {}^6_n {}^{13}_n + \bar{x}_2 \bar{x}_4 \bar{x}_6 {}^2_n {}^4_n {}^6_n {}^{13}_n + \bar{x}_8 \bar{x}_9 {}^8_n {}^9_n + \bar{x}_1 \bar{x}_3 \bar{x}_9 {}^1_n {}^3_n {}^9_n {}^{14}_n$$
$$+ \bar{x}_1 \bar{x}_4 \bar{x}_9 {}^1_n {}^4_n {}^9_n {}^{14}_n + \bar{x}_2 \bar{x}_3 \bar{x}_9 {}^2_n {}^3_n {}^9_n {}^{14}_n + \bar{x}_2 \bar{x}_4 \bar{x}_9 {}^2_n {}^4_n {}^9_n {}^{14}_n + \bar{x}_8 \bar{x}_{10} {}^8_n {}^{10}_n$$
$$+ \bar{x}_1 \bar{x}_3 \bar{x}_{10} {}^1_n {}^3_n {}^{10}_n {}^{14}_n + \bar{x}_1 \bar{x}_4 \bar{x}_{10} {}^1_n {}^4_n {}^{10}_n {}^{14}_n + \bar{x}_2 \bar{x}_3 \bar{x}_{10} {}^2_n {}^3_n {}^{10}_n {}^{14}_n$$
$$+ \bar{x}_2 \bar{x}_4 \bar{x}_{10} {}^2_n {}^4_n {}^{10}_n {}^{14}_n$$

From the above equation, the x_j list has 27 terms after eliminating the identical terms, and then the generated $\hat{x}_j$ list will have 60 terms after all elimination. To find a cover for the 60 terms may present some problems. A minimal set of 9 tests can be found to cover these terms. Hence, the abnormal true test set has 9 tests. Then, another cover (1st,7th,11th and 20th term in F^1) must be found to cover all the checkpoint.

For those 4 terms, we find the exclusive normal input causes:

for $\bar{x}_5\bar{x}_7$, $x_i = x_1x_2\bar{x}_5\bar{x}_7x_8 + x_1x_2\bar{x}_5\bar{x}_7x_9x_{10} + x_3x_4\bar{x}_5\bar{x}_7x_8 + x_3x_4\bar{x}_5\bar{x}_7x_9x_{10}$;

for $\bar{x}_1\bar{x}_3\bar{x}_6$, $x_i = \bar{x}_1x_2\bar{x}_3x_4x_5\bar{x}_6x_7x_9x_{10}$;

for $\bar{x}_8\bar{x}_9$, $x_i = x_1x_2x_5x_6\bar{x}_8\bar{x}_9x_{10} + x_1x_2x_7\bar{x}_8\bar{x}_9x_{10} + x_3x_4x_5x_6\bar{x}_8\bar{x}_9x_{10}$
$+ x_3x_4x_7\bar{x}_8\bar{x}_9x_{10}$;

for $\bar{x}_2\bar{x}_4\bar{x}_{10}$, $x_i = x_1\bar{x}_2x_3\bar{x}_4x_5x_6x_8x_9\bar{x}_{10}$.

Therefore another 4 tests are needed to verify all the checkpoints. Altogether, 9 + 4 = 13 tests are required to form an MFTS, which is more than double the number which is necessary.

If we had started with the $\bar{F}^1$ equation to find the abnormal false test set, we would have needed 24 tests.

The above discussion shows that our algorithm is superior to these other algorithms in everyone of the criteria for a good algorithm listed earlier.

In the next chapter, we will explore some ideas on how to design a network such that it has the smallest MFTS and is easy to diagnose.

CHAPTER 5

DESIGN PRINCIPLES

5.1. Introduction

In the theory of logical design, much attention has been given to the design of networks which are in some sense optimal. The objective is to build a network with minimum cost. The classical cost factor is proportional to the number of gates and, sometimes, lines between gates. To design a network with minimum 2-level AND-OR networks, the Quine-McCluskey [22] algorithm has been widely used. If the number of variables is less than seven, K-maps can be used efficiently to design a two level AND-OR network. Various algorithms or programs exist for designing a circuit with a minimum number of gates under somewhat different constraints; Hellerman [23] has cataloged a set of minimal NAND and NOR realizations of all functions of 3-variables, while Ninomiya [24] has formed a similar catalog of all 4-variable functions.

As integrated circuit technology advances, the cost of gates is being reduced significantly. Hence, the criterion of designing a network with a minimum number of gates is becoming less important. On the other hand, the cost of generating diagnostic tests and running the diagnostic tests is going up considerably. We consider that a network has minimum cost if throughout the whole life of the network the total cost, which includes design effort, manufacturing cost and maintenance cost, is minimized. In this chapter, we will focus our attention on reducing the maintenance cost.

5.2. Some Previous Work on Design Principles

Metze and his former students have long been advocating designing networks which are easily diagnosable. This will reduce maintenance costs. Among their papers, one of the most important is by Schertz and Metze [10].

They have proved the SFTS for a single irredundant restricted fanout-free network with primary input fanout allowed is also an MFTS. Therefore, if we can design a network in this configuration, generating the MFTS for it is the same as generating the SFTS. A different proof for this result was given in Chapter 4. Naturally, we are going to ask what other kinds of networks will have the same property. Unfortunately, there are none, as we can see from Example 3.3, which is the simplest deviation from the restricted fanout-free network, and which has an SFTS which is not an MFTS. Therefore, this example prevents us from going further in this direction.

But, as we have shown in Chapter 3, if the right algorithm is chosen, generating an MFTS for any irredundant network is really not difficult at all. In some sense, it is even simpler, since the number of prime faults is usually less than the number of single faults.

However, applying our algorithm to some circuits is easier than applying it to others. Why is this so? How should we design networks that are easily diagnosed, and hence, have a lower cost of generating test sets? We will discuss these points in later sections.

5.3. Bounds on the MFTS for the Algorithm

Before we continue our discussion on design rules, we shall find the bounds on the MFTS for the algorithm in Chapter 3.

For an irredundant network, the algorithm first finds a set of prime faults PF. Then the tests for those prime faults are derived and their associated masking faults are determined. Once a prime fault is detected by some generated test, we will not generate another test to detect it unless the fault is involved in a cycle. Then, we generate a new test to replace the old test, or we add a new test if the old test is needed because it detects another prime fault. This leads to the following theorem.

Theorem 5.1: The number of tests in the MFTS generated by our algorithm for an irredundant network is bounded by the number of prime faults in that network.

Proof: From our algorithm, we pick a prime fault only once and generate a test which detects that particular prime fault and determines its corresponding masking faults. The test may detect other prime faults, but every test detects at least one particular prime fault. Therefore, after all the tests that cover all the prime faults have been generated, the number of tests will be bounded by the number of prime faults. If the test set generated passes the verification, namely it is an MFTS, we are done.

If the test set does not pass the verification, then we must have a loop. Assume the loop contains prime faults $f_1, f_2, \ldots, f_L$. If the network is irredundant, any such loop can be broken. Therefore, we can

find another test t_i', $1 \leq i \leq L$, that detects f_i with a different masking fault from t_i. Then the test set plus t_i' will form a new test set. If the new test set is not an MFTS, the procedure is repeated. If t_i detects f_i only, we can replace t_i by t_i' in our test set, and the new test set is an MFTS with the same number of tests as our old test set. So the MFTS is still bounded by the number of prime faults.

If t_i detects other prime faults besides f_i, then we add the new test t_i' to the first generated test set forming a new test set, which is still bounded by the number of prime faults. Hence, $|MFTS| \leq |PF|$.

Q.E.D.

Example 5.1: This circuit, shown in Figure 5.1, has $|MFTS| = |PF|$. Any irredundant cascade circuit has this property, as every test can detect only one prime fault. Also, every test detects that prime fault completely. This example shows our upper bound is a least upper bound.

There also exists a lower bound for $|MFTS|$.

Theorem 5.2: Let n_1 and n_o be the maximal number of inputs that are associated with prime faults of a gate at odd and even levels, respectively. Then, $n_1+n_o \leq |MFTS|$.

Proof: Let G_1 and G_o be the gates that have n_1 and n_o inputs, respectively. Any test that detects one input line stuck-at-i for G_1 or G_o cannot detect the co-input lines stuck-at-i for G_1 or G_o. Therefore, in order to make an MFTS, we need at least n_1+n_o tests. Q.E.D.

Example 5.2: The circuit shown in Figure 5.3b has $|MFTS| = n_1+n_o$. (Here $n_1 = 4$, $n_o = 2$.) This example shows the lower bound is the greatest lower bound.

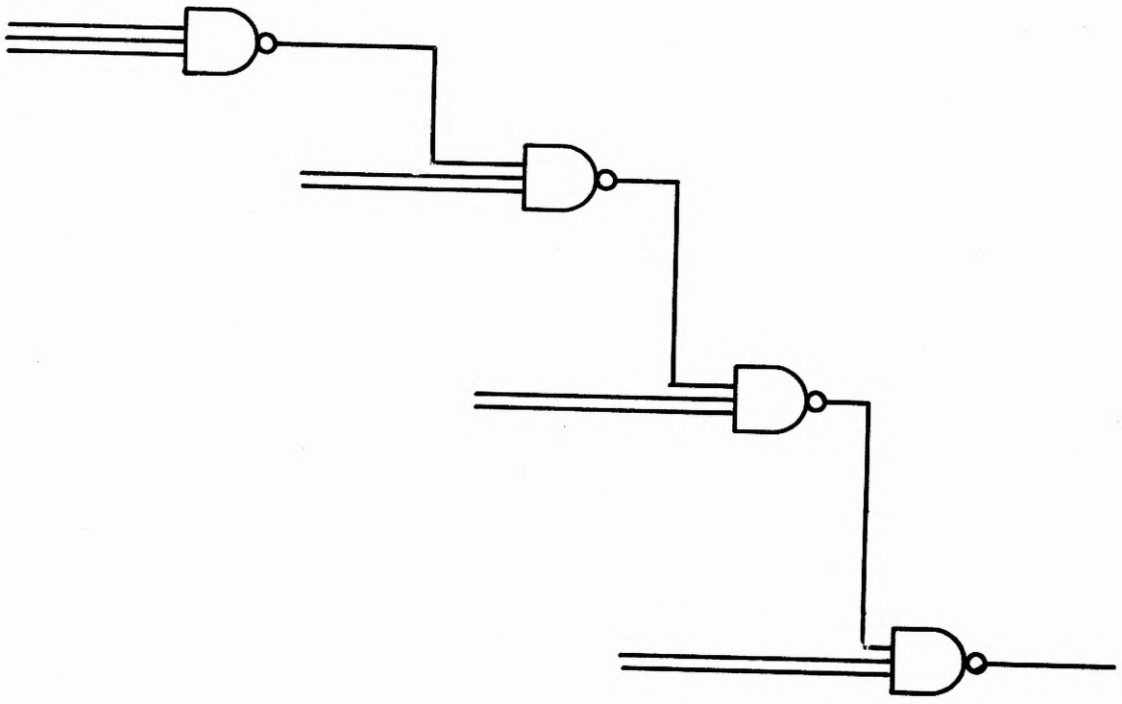


Figure 5.1. A cascade of NAND network that has $|MFTS| = |PF|$.

Corollary: For an irredundant network, we have $n_1+n_o \leq |MFTS| \leq |PF|$.

Proof: Follows directly from Theorems 5.1 and 5.2.

Finally, we notice that for any single gate, the lower bound equals the upper bound. This is an extreme case.

5.4. A New Design Rule

It was first pointed out by Friedman [17] that redundancy is incompatible with diagnosis. Faults in redundant circuits can mask the original detectable faults under a certain test set, and they become undetectable. If a circuit has redundancy, there does not exist any MFTS for that network. In other words, there is a large number of faults that are undetectable. Therefore, redundancy should be eliminated from the circuit. Fortunately, the algorithm in Chapter 3 can identify any redundancy. Undesirable redundancy can therefore be eliminated before production, such that every network will be diagnosable.

Furthermore, in order to reduce production costs, it should be easy to generate the MFTS, and in order to reduce maintenance cost the number of tests in the MFTS should be as small as possible.

From the algorithm, we can see that fanout-free circuits are the easiest to diagnose. Unfortunately, most circuits are not fanout-free. However, since each fanout makes generating the MFTS more difficult, we should try to reduce their number.

Also, we want to minimize the size of the MFTS. As we proved in Section 5.3, $|MFTS|$ is bounded by n_1+n_o and $|PF|$. To minimize the number of prime faults will certainly reduce the upper bound. As prime

faults are related to primary input lines, fanout branch lines and output lines of prime gates, we should try to minimize them. Primary input lines usually cannot be reduced for a given network when realizing a given function. But, we can still minimize the fanouts.

From the above discussion, we can make the following conjecture.

If a network for a given function has a minimum number of fanouts, then it tends to have the least number of diagnostic tests in its MFTS. In addition, it is easier to generate the MFTS.

Besides minimizing the number of fanouts, we should try to minimize the sum of n_1 and n_o, that is, the maximum number of inputs that are associated with prime faults, of one gate at an odd and one at an even level of the network. This goal is another subject for research.

5.5. Examples

We give some examples to illustrate the above conjecture.

Example 5.3: The Boolean function $f = \bar{x}_1\bar{x}_3 + \bar{x}_2\bar{x}_4 + \bar{x}_3\bar{x}_4 + x_1x_2x_3$ has 6 different irredundant network configurations, shown in Figures 5.2a to 5.2f. The MFTS for each of these networks is given in Table 5.1.

From Table 5.1, we can see Figure 5.2d has the minimum number of fanouts, and the minimum number of prime faults. Hence, it also has the minimum number of tests in its MFTS. From a diagnostic point of view, we should use this network configuration as our choice, as it will be the most economical to build and to maintain during its lifetime. Traditionally designers tend to choose the network in Figure 5.2e, because it has the minimum number of gates and lines.

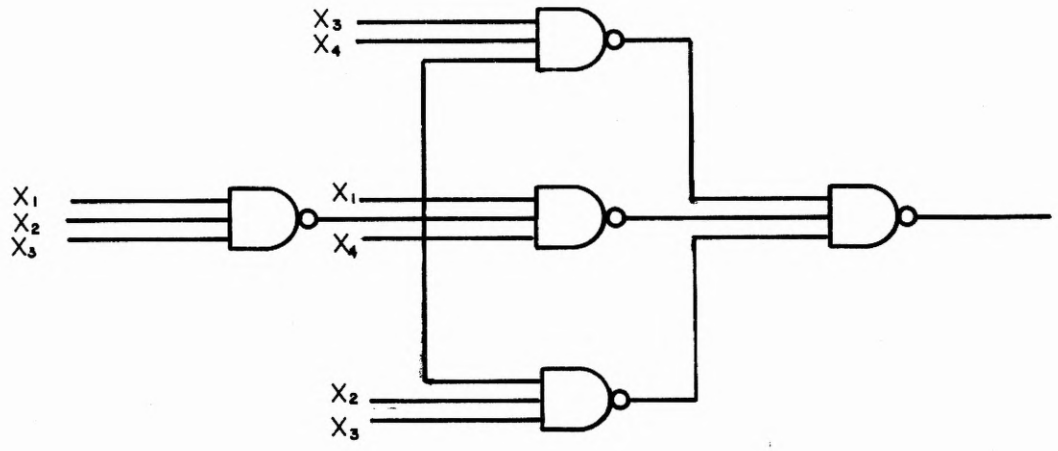


Figure 5.2 a)

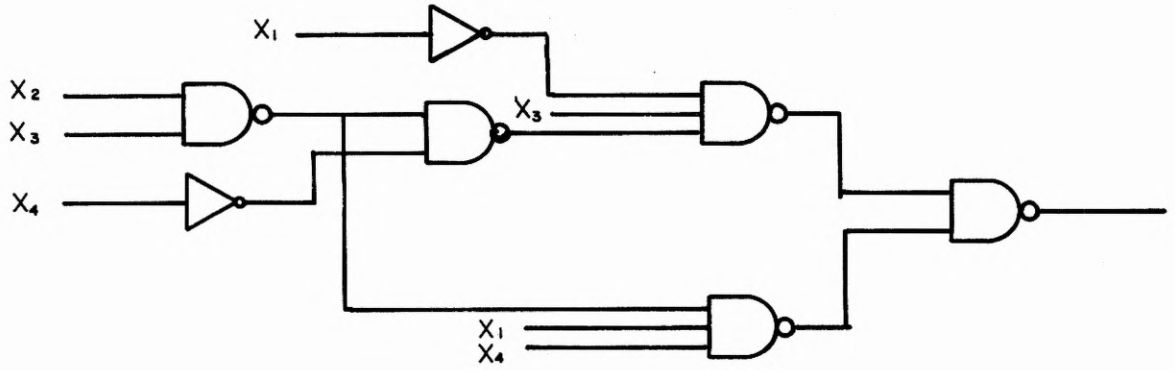


Figure 5.2 b)

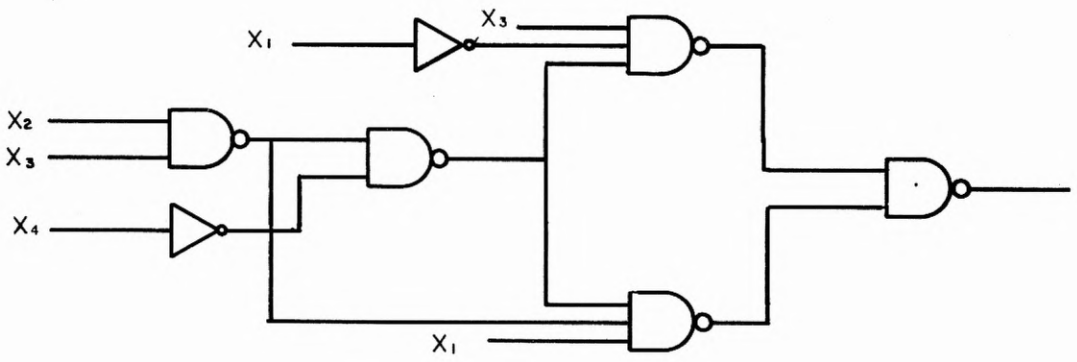


Figure 5.2 c)

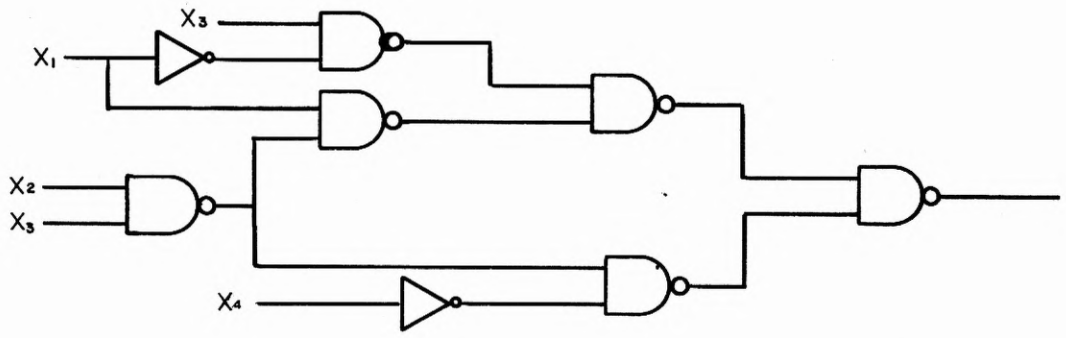


Figure 5.2 d)

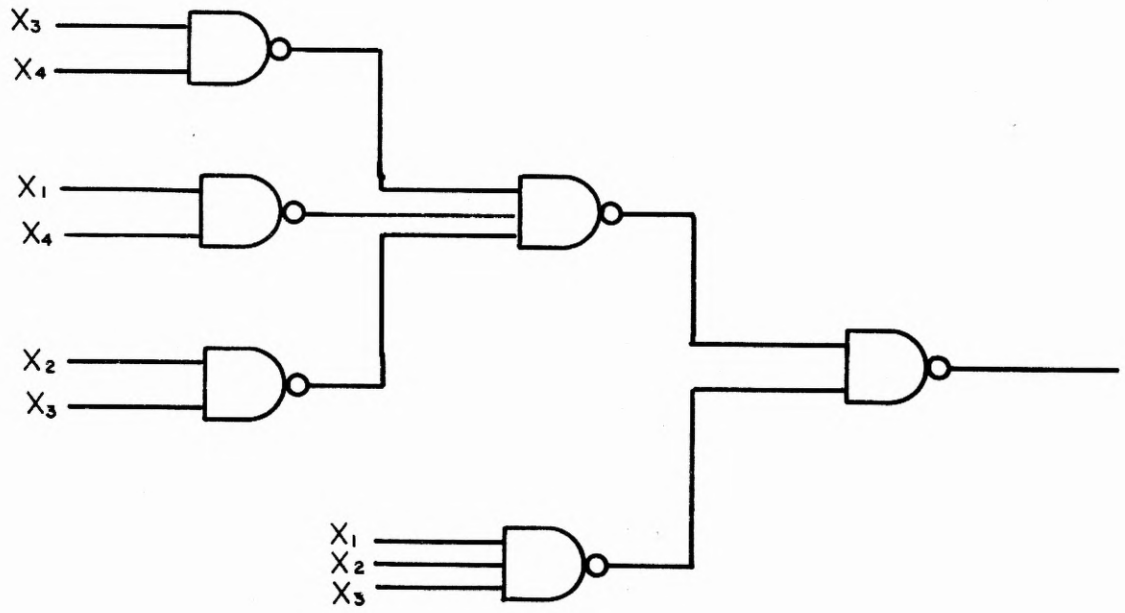


Figure 5.2 e)

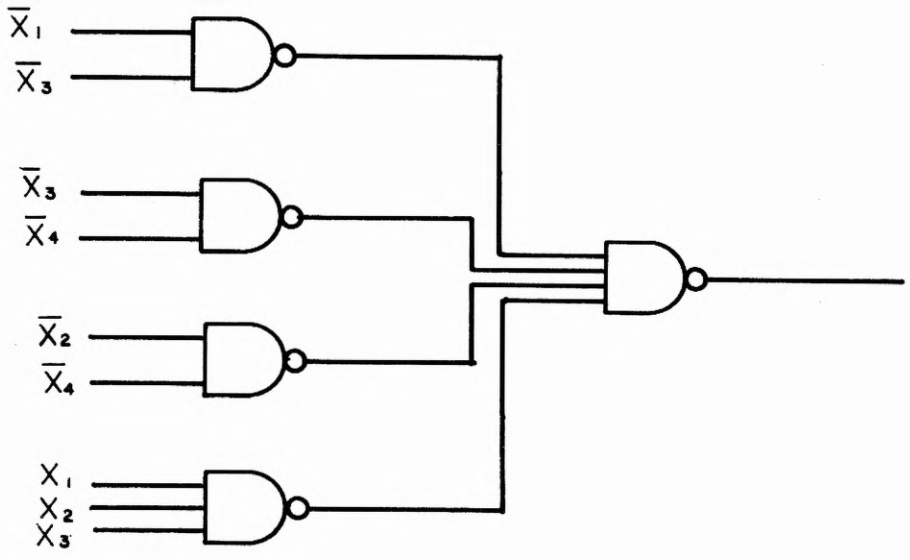


Figure 5.2 f)

Figure numbers in Fig. 5.2	a	b	c	d	e	f
number of gates	6	8	8	8	6	9
number of lines	16	15	15	14	14	18
number of fanouts	5	4	4	3	4	5
number of prime faults	15	11	12	11	13	13
number of tests for MFTS	7	7	6	5	7	7

Table 5.1

But, it has the maximum number of tests in its MFTS. Thus, it is harder to generate the MFTS. Since |MFTS| is larger, it also takes a longer time to diagnose the circuit. In the long run, this is probably a very poor choice.

Example 5.4: This is another example to illustrate that a circuit with fewer fanouts ends up with fewer tests in its MFTS. In this case, Figure 5.3b has 4 fanouts and |MFTS| = 6, whereas Figure 5.3a has 6 fanouts and |MFTS| = 10, even though it has a minimum number of gates.

In addition to these small examples, the above guidelines have been tried on a much larger sample and they have yielded the same result. Furthermore, every network that has a minimum number of gates invariably yields the largest |MFTS|. Therefore, from a diagnostic point of view, we should not design networks with a minimum number of gates, but, with a minimum number of fanouts.

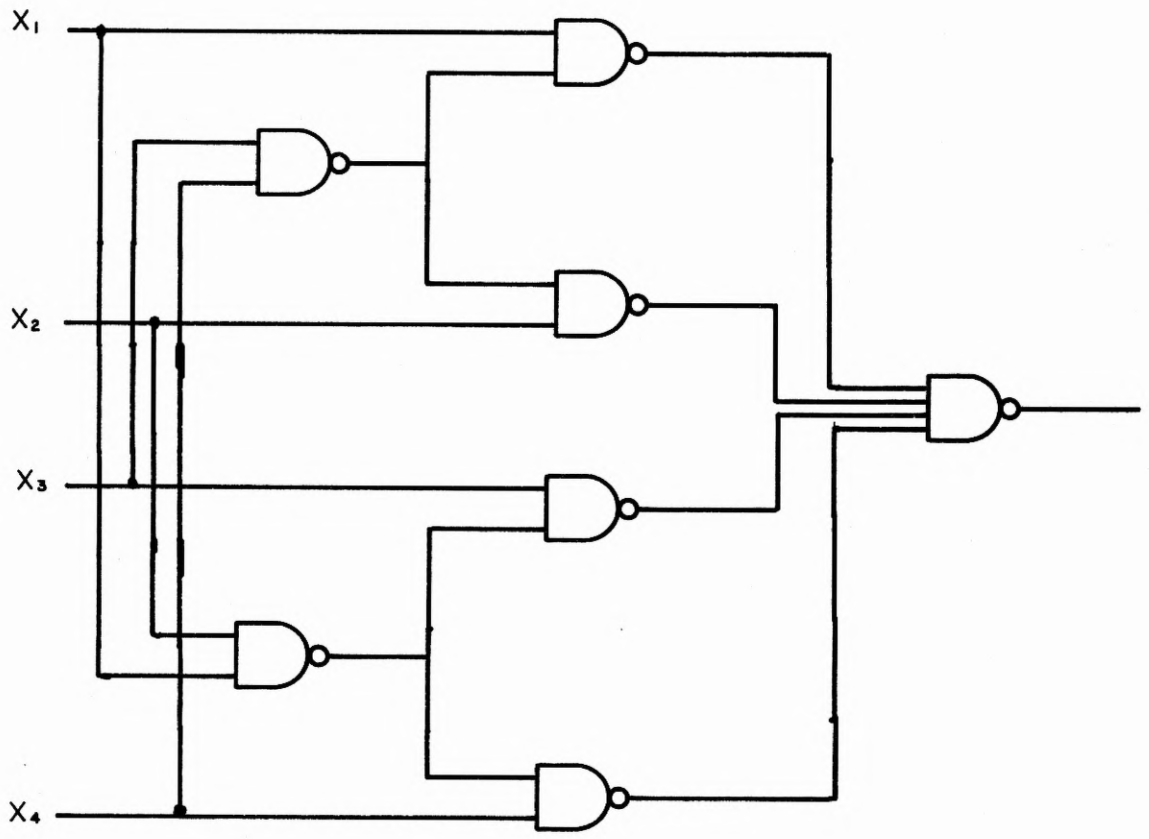


Figure 5.3a. Realization of $F = \bar{x}_1 x_2 + \bar{x}_3 x_4 + x_1 \bar{x}_4 + x_3 \bar{x}_2$ with a minimal number of gates.

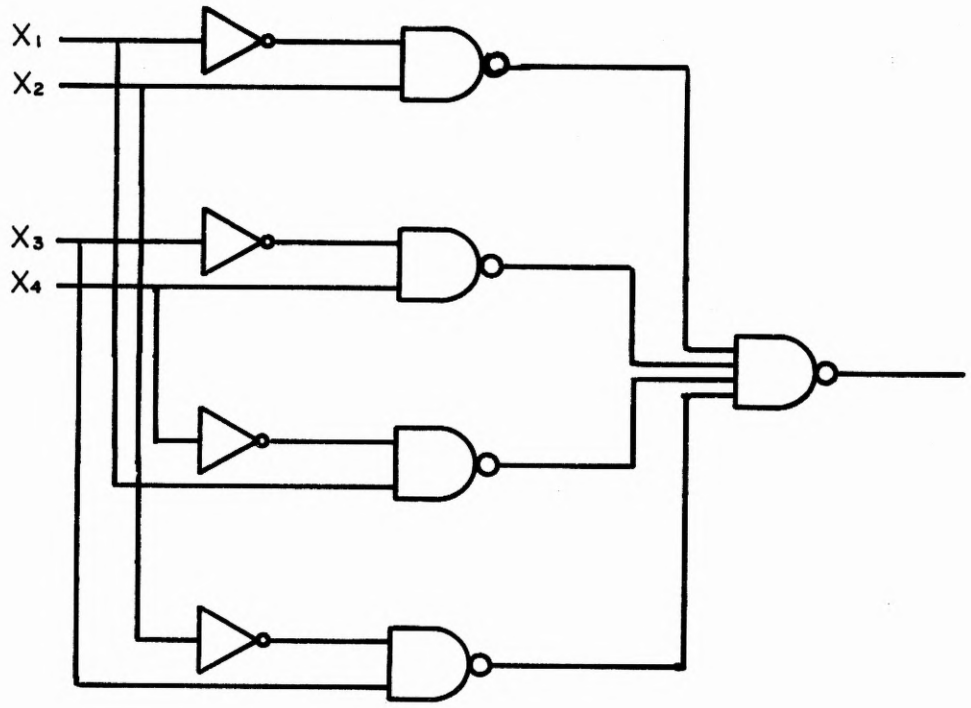


Figure 5.3b. Realization of $F = \bar{x}_1 x_2 + \bar{x}_3 x_4 + x_1 \bar{x}_4 + x_3 \bar{x}_2$ with a minimum number of fanouts.

CHAPTER 6

CONCLUDING REMARKS

6.1. The Idea Behind the Algorithm

Because of the large number of multiple faults even in a small circuit, their study has been avoided until recently. After fault collapsing and fault equivalence concepts were introduced in the late Sixties, fault analysis was simplified greatly, since individual equivalence classes can be analyzed instead of individual faults. To find all functional equivalence classes of single faults is not easy; to find all the functional equivalence classes for multiple faults is even more difficult. The "prime fault" concept introduced here reduces the complexity of the problem. Any multiple fault can be represented as a combination of one or more prime faults. Thus, the problem of finding multiple fault equivalence classes is solved.

For a circuit with n prime faults, there are at most 2^n distinct fault equivalence classes. That number is still very large for even small n. This discourages the generation of tests for every individual class. However, it is not difficult to find tests that detect specific prime faults and also detect several other multiple fault classes, although it is then necessary to identify those multiple faults that the test does not detect. This leads to the definition of a masking fault for a given fault under a given test.

With the concept of a prime fault and a masking fault and the discovery that finding a set of tests that is an MFDTS is very similar to finding a cover for a table, our algorithm was born.

6.2. A Summary of the Thesis

An efficient algorithm to generate a multiple fault detection test set for a single output combinational network has been developed. The main features of the algorithm are as follows.

The first feature of the algorithm is that it is not necessary to sensitize more than one path deliberately in order to detect a certain fault. This eliminates tedious trial and error procedures like Roth's D-algorithm for faults whose detection requires more than one path to be sensitized.

A second feature is the use of a consistency step to generate a test that is locally optimal, i.e. it detects as many faults as possible with the fewest masking faults. In addition, the number of tests can be reduced by choosing a different path each time. Usually, an optimal MFDTS can be obtained.

A third feature of the algorithm is that any multiple-line redundancies in a circuit can be identified during the process of generating its MFDTS. This will enable a designer to remove any undesired redundancy before final production.

Besides generating the MFDTS of a circuit, the algorithm can also be used to list the faults a given test or set of tests detects or to determine whether a given set of tests is an MFDTS or not. The algorithm can easily be generalized to multiple-output combinational networks.

Experience with the algorithm was used to suggest a design principle, namely to design networks with a minimum number of fanouts

in order to simplify the generation of the MFDTS of the network and to minimize the size of the MFDTS.

6.3. Further Research

An immediate suggestion is the implementation of the algorithm as a computer program. Even though the algorithm can be applied to small circuits by hand, it is more convenient and time saving to use computer programs for big combinational circuits and for the routine analysis of examples.

There is an open question concerning the identification of redundancies. After a set of tests to detect all the prime faults is generated, we have to verify whether the set of tests is an MFDTS. If it is not, there is a loop of masking faults and other tests have to be examined in an attempt to break the loop. If the loop cannot be broken, the circuit contains a redundancy. It would seem that in generating the test set according to the algorithm, the appearance of a loop probably implies redundancy in the circuit. In other words, we conjecture that it would be futile to try to break the loop. If this conjecture can be proved it would greatly simplify the identification of redundancies in a circuit.

The implementation of the design rule proposed in Chapter 5, to use a minimum number of fanouts and prime fault lines, so that a network that has the minimum number of tests in its MFDTS and is also easy to diagnose is obtained, should also be explored. This implementation would require a method to realize a given Boolean equation with a minimum number of fanouts.

Another interesting problem is so-called module diagnosis. Suppose we have small circuits $c_1, c_2, \ldots, c_n$ such that c_i is diagnosable, i.e. every c_i has a corresponding $MFDTS_i$. Now, if we connect those c_i in any arbitrary manner to form a large circuit C, can we find the MFDTS for C directly from the $MFDTS_i$?

Last but not least, how can one extend our algorithm to general sequential circuits? Since sequential circuits can be considered as combinational circuits plus feedback lines, a given test can be chosen such that every feedback line is desensitized or blocked by other values. Then, the test will detect the same kinds of faults with the same masking faults as if the circuit were a combinational circuit. Therefore, the basic idea of our algorithm may be applied at least to synchronous sequential circuits. The details, however, remain to be worked out.

REFERENCES

1. Kautz, W. H., "Fault testing and diagnosis in combinational digital circuits," IEEE Trans. on Computers, Vol. C-17, pp. 352-366, April 1968.

2. Poage, J. F., "Derivation of optimum tests to detect faults in combinational circuits," Math. Theory of Automata, Polytechnic Press, Brooklyn, 1963.

3. Chang, H. Y., "An algorithm for selecting an optimum set of diagnostic tests," IEEE Trans. on Electronic Computers, Vol. EC-14, pp. 706-711, October 1965.

4. Powell, T. J., "A module diagnostic procedure for combinational logic," C.S.L. Report R-413, University of Illinois, Urbana, April 1969.

5. Armstrong, D. B., "On finding a nearly minimal set of fault detection tests for combinational logic networks," IEEE Trans. on Electronic Computers, Vol. EC-15, pp. 66-73, February 1966.

6. Sellers, F. F., Hsiao, M-Y, and Bearnson, L. W., "Analyzing errors with the Boolean difference," IEEE Trans. on Computers, pp. 676-683, July 1968.

7. Roth, J. P., et al., "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," IEEE Trans. on Computers, pp. 567-579, October 1967.

8. Poage, J. F. and McCluskey, E. J., "Derivation of optimum test sequences for sequential machines," Proc. 5th Annual Symposium on Switching Theory and Logical Design, pp. 121-132, October 1964.

9. Schertz, D. R. and Metze, G., "A new representation for faults in combinational digital circuits," IEEE Trans. on Computers, pp. 858-866, August 1972.

10. Schertz, D. R. and Metze, G., "On the design of multiple fault diagnosable networks," IEEE Trans. on Computers, pp. 1361-1364, November 1971.

11. Hayes, J. P., "A NAND model for fault diagnosis in combinational logic networks," IEEE Trans. on Computers, pp. 1496-1506, November 1971.

12. Yau, S. S. and Tang, Y. S., "An efficient algorithm for generating complete test sets for combinational logic circuits," IEEE Trans. on Computers, pp. 1245-1251, November 1971.

13. Bossen, D. C. and Hong, S. J., "Cause-effect analysis for multiple fault detection in combinational networks," IEEE Trans. on Computers, pp. 1252-1257, November 1971.

14. Clegg, F. W. and McCluskey, E. J., "Fault equivalence in combinational logic networks," IEEE Trans. on Computers, pp. 1286-1293, November 1971.

15. Schertz, D. R., "On the representation of digital faults," C.S.L. Report R-418, University of Illinois, Urbana, May 1969.

16. Hayes, J. P., "A study of digital network structure and its relation to fault diagnosis," C.S.L. Report R-467, University of Illinois, Urbana, May 1970.

17. Friedman, A. D., "Fault detection in redundant circuits," IEEE Trans. on Computers, pp. 99-100, February 1967.

18. Gimpel, J. F., "The minimization of TANT networks," IEEE Trans. on Computers, pp. 18-38, February 1967.

19. Davidson, E. S., "An algorithm for NAND decomposition under network constraints," IEEE Trans. on Computers, pp. 1098-1109, December 1969.

20. Dandapani, R., Reddy, S. M., and Robinson, J. P., "An investigation into redundancy and testability of combinational logic networks," Technical Report No. 32, Themis Project, University of Iowa, September 1970.

21. Lee, S. H. and Davidson, E. S., "Redundancy testing in combinational networks," Technical Report No. 67, D.S.L., Stanford University, May 1973.

22. McCluskey, E. J., "Introduction to the theory of switching circuits," McGraw-Hill, New York, 1965.

23. Hellerman, L., "A catalog of three variable OR-invert and AND-invert logical circuits," IEEE Trans. on Electronic Computers, pp. 198-223, June 1963.

24. Ninomiya, I., "A study of the structures of Boolean functions and its application to the analysis of switching circuits," Memoirs of the faculty of engineering, Nagoya University, Vol. 13, pp. 149-363, 1961.

25. Kohavi, I. and Kohavi, Z., "Detection of multiple faults in combinational logic networks," IEEE Trans. on Computers, pp. 556-568, June 1972.

26. Chang, H. Y., Manning, E. G., and Metze, G., "Fault diagnosis of digital systems," Wiley Interscience, 1970.

27. Friedman, A. D. and Menon, P. R., "Fault detection in digital circuits," Prentice-Hall, 1971.

28. Sellers, F. F., Hsiao, M-Y, and Bearnson, L. W., "Error detecting logic for digital computers," McGraw-Hill, 1968.

VITA

Charles Wei-Yuan Cha was born in Shanghai, China on September 3, 1945. He received a B.S. degree from New Asia College at the Chinese University of Hong Kong with Cum Laude in 1968. He entered the University of Illinois and received an M.S. degree from the Department of Mathematics in 1970. Then he entered the Department of Computer Science, University of Illinois where he held a teaching assistantship and research assistantship. He joined the Digital Systems Group at the Coordinated Science Laboratory of the University of Illinois in 1972, where he held a research assistantship until 1974.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

MULTIPLE FAULT DIAGNOSIS IN COMBINATIONAL NETWORKS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(First name, middle initial, last name)*

Charles Wei-Yuan Cha

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June, 1974	114	28

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAAB07-72-C-0259	R-650
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	UILU-ENG 74-2215

10. DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Joint Services Electronics Program through U.S. Army Electronics Command, Fort Monmouth, New Jersey

13. ABSTRACT

A new concept, the prime fault, is introduced for the study of multiple fault diagnosis in combinational logic networks. It is shown that every multiple fault in a network can be represented by a funcationally equivalent fault with prime faults as its only components. The use of prime faults greatly simplifies multiple fault analysis and test generation.

DD FORM 1 NOV 65 1473

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Multiple Fault Diagnosis						
Prime Faults						
Design Rule						