April 16, 1973

To the Graduate Council:

I am submitting herewith a thesis written by Thomas Earl Akin entitled "Computer Implemented Minimization of Fault Detection Experiments for Combinational Circuits." I recommend that it be accepted for nine quarter hours of credit in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert E. Bartlebaugh
Major Professor

We have read this thesis and recommend its acceptance:

Robert M. Aiken

Sara R. Jordan

Accepted for the Council:

Hilton A. Smith
Vice Chancellor for
Graduate Studies and Research

COMPUTER IMPLEMENTED MINIMIZATION OF FAULT DETECTION EXPERIMENTS FOR COMBINATIONAL CIRCUITS

A Thesis

Presented to

the Graduate Council of

The University of Tennessee

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Thomas Earl Akin

June 1973

ACKNOWLEDGMENTS

The author takes this opportunity to express his sincere gratitude to Dr. Robert E. Bodenheimer for his suggestion of the topic and for his advice and encouragement during the preparation of this thesis. Also, the author wishes to express his appreciation to Dr. Robert M. Aiken and Dr. Sara R. Jordan for their services on the Master's degree committee. Finally, the author is deeply indebted to his parents, Mr. and Mrs. T. R. Akin, for their encouragement and assistance which has made the computer science education both possible and worthwhile.

1089921

ABSTRACT

This thesis describes two basic methods for generating fault-detection tests in single-output, multiple-level combinational circuits. The two methods are: (1) table of combinations method; and (2) Poage's method. An illustrative example is presented with each description.

The circuits studied were realized using AND, OR, NOT, NOR, and NAND logic elements. The formation, properties, and uses of the fault table are discussed.

Finally, an IBM System 360/65 FORTRAN IV and ALC digital computer program is presented. The basis of this program is the table of combinations method. The program generates the fault table and finds a minimal cover for the fault table. This minimal cover represents a minimal fault-detection experiment. The program can handle from one to one hundred gate circuits and two to seven input variables. Individual gate restrictions are a maximum of eight inputs and a maximum fanout of ten. This research was carried on under the single-fault assumption for irredundant combinational circuits.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	Statement of Problem	2
	Scope of Investigation	3
II.	FAULTS IN COMBINATIONAL CIRCUITS	10
III.	FAULT-DETECTION TEST GENERATION	17
	Table of Combinations Method	17
	Poage's Method	24
IV.	THE FAULT TABLE	29
	Essential Tests	30
	Row Dominance	30
	Column Dominance	31
	Cyclic Table	32
	Minimal Fault-Detection Experiment	34
V.	MINIMIZATION WITH THE AID OF A DIGITAL COMPUTER	39
VI.	SUMMARY AND CONCLUSIONS	51
LIST OF REFERENCES		53
APPENDIXES		55
A.	LISTING OF COMPUTER PROGRAM	56
B.	SAMPLE INPUTS AND OUTPUTS	87
VITA		93

LIST OF FIGURES

FIGURE		PAGE
1.	Block Diagram of Combinational Circuit	4
2.	Symbols for Logical Elements	4
3.	Two Realizations for AB' + A'B	6
4.	Typical Single-Output, Multiple-Level Circuit . .	8
5.	Gate Behavior When a Gate Input Is s-a-0	12
6.	Gate Behavior When a Gate Input Is s-a-1	13
7.	Normal and Faulty Circuit Behavior	14
8.	A Circuit and Its Fault Table Generated by Table of Combinations Method	19
9.	A Circuit and Its Fault Table Generated by Poage's Method	26
10.	Minimization by the Branching Method	33
11.	Circuit and Its Fault Table	36
12.	Minimization of Fault Table	37
13.	A Circuit and Its Input Data	41
14.	Two Examples of Different Numbering of Gates in a Circuit	43
15.	General Flowchart for the Program	47
16.	Control Card Sequence for Source Deck	49
17.	Control Card Sequence for Object Deck	50

CHAPTER I

INTRODUCTION

Switching circuits are used in the design and construction of computers and many other electronic devices. In the use of these circuits, maximum reliability is of the utmost importance. With the growing use of the digital computer in all areas of science and industry, the topic of digital reliability has begun to receive more attention. This branch of automata theory is an increasingly important area of Computer Science.

The reliability of a digital system depends on its components, which may either function correctly, or may alternatively malfunction in any one of the possible input modes. The malfunction of a hardware component is called a fault. Component faults are assumed to be permanent as compared to intermittent faults which may change their nature during execution and testing (1).

One approach to the aspect of maintaining high reliability is to replace the failed component. This technique requires that first, the fault be detected and second, the failed component be located within the circuit. Another approach is to use some type of circuit redundancy which intentionally masks the failed component from affecting the

circuit output. The approach used is dependent upon the designer's objective and the time required to detect and locate the faulty component as compared to the extra cost of redundant hardware.

The minimization of these permanent faults either by replacing or masking the failed components is the most important element in the design of a digital computing system.

I. STATEMENT OF PROBLEM

The outputs of a circuit containing a failed component will be different from their normal values for some input combinations applied to the circuit. Any such input combination which produces an incorrect output is a test for detecting that fault because the presence of the fault can be detected by applying this input combination and observing the output. In general, a particular input combination may be a test for detecting several faults in a circuit and there may be several tests for detecting a given fualt (2).

Any irredundant circuit with n inputs can be completely tested by applying all 2^n input combinations to the circuit. As n becomes large, this procedure becomes undesirable due to the large number of input combinations to be applied in order to detect all faults. A smaller set of tests is often sufficient for detecting all faults that are likely to occur. This set of fault-detection tests is called a fault-detection

experiment (1). Thus, one of the important problems associated with the testing of large combinational circuits is that of obtaining a minimal fault-detection experiment. This thesis proposes to present a solution to this problem.

The importance of obtaining a minimal fault-detection experiment is not difficult to state. Reduction in the number of input combinations to be applied in order to detect all faults can save time in the detection, location and repair or replacement of a faulty component which in turn leads to increased circuit reliability.

II. SCOPE OF INVESTIGATION

The objective of the research presented herein is the minimization of fault-detection experiments, a topic from the many available in the field of Computer Science. Of the two types of circuits normally used in a digital computer, the author has chosen to deal with single-output combinational circuits.

Combinational circuits are used to realize combinational functions where the output is dependent only on the present values of the inputs and are characterized by the absence of feedback as shown in Figure 1.

Several different symbols for the basic logic element are found in the design of circuits today. Figure 2 shows the set of symbols for the AND, OR, NOT, NOR and NAND elements which were selected for use in this research.

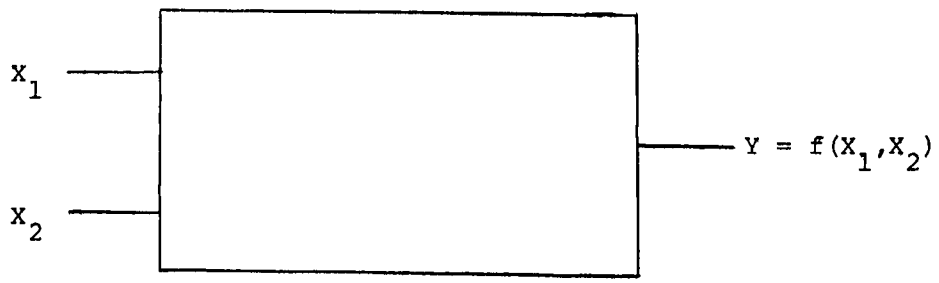
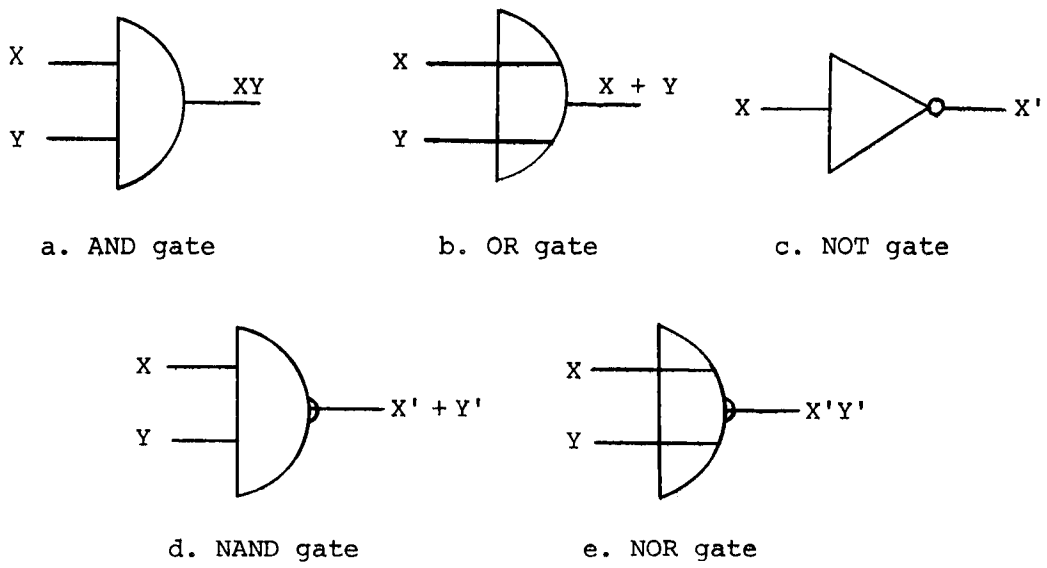


Figure 1. Block diagram of combinational circuit.



a. AND gate b. OR gate c. NOT gate

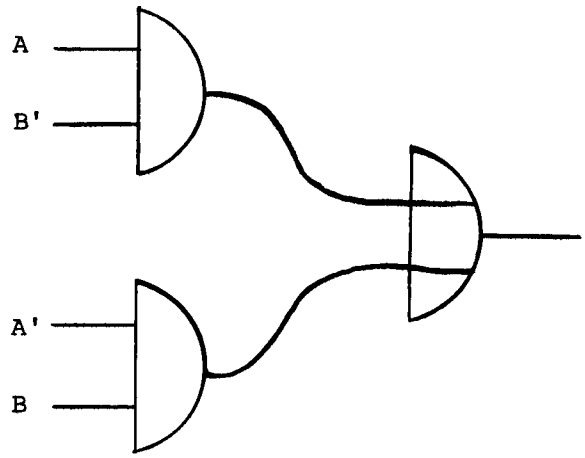
d. NAND gate e. NOR gate

Figure 2. Symbols for logical elements.

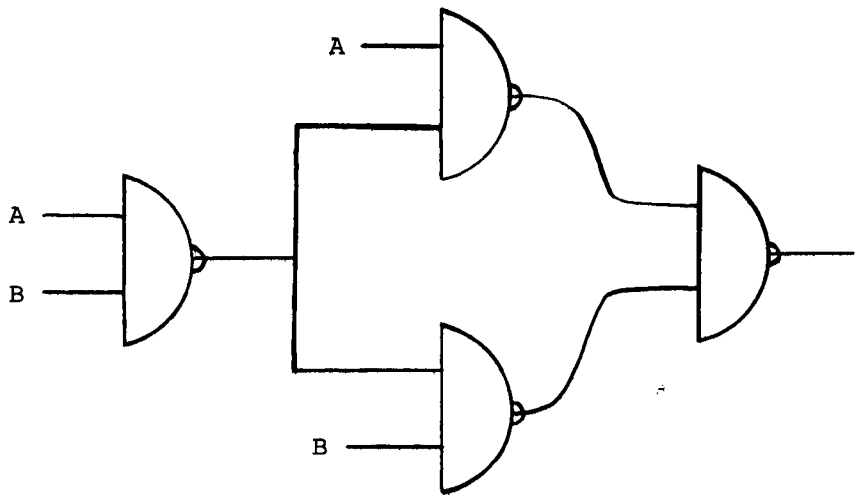
Permanent faults caused by hardware failure that can occur in combinational circuits are discussed in Chapter II. Also presented in this chapter is the relationship between detectable faults and irredundant circuits and the relationship between undetectable faults and redundant circuits.

Two formal procedures for generating fault-detection tests are presented in Chapter III. The table of combinations method for single faults and Poage's method for both single and multiple faults are introduced. This leads to the fault table, its reduction and finally to a minimal fault-detection experiment which are discussed in Chapter IV.

A digital computer program, written by the author, is explained in Chapter V. This program accepts as inputs a particular logical gate configuration for a combinational function and implements the table of combinations method to generate the complete fault table for the circuit. This type of input is used over the canonical representation of a combinational function because any given combinational function may be realized by more than one logical gate configuration (3). This is shown in Figure 3. The program then calculates a minimal fault-detection experiment by employing the reduction methods for the fault table as explained in Chapter IV. The limitations in regard to the type and size of circuit which the program can handle are also explained in Chapter V.



a. AB' + A'B



b. AB' + A'B

Figure 3. Two realizations for AB' + A'B.

The use of the digital computer as a tool in the improved reliability and study of logical circuitry has increased markedly in recent years (3,4). As shown in Chapter V, the computer can provide optimum solutions in a short time to vastly complex reliability problems that formerly required the expenditure of large amounts of human effort and time.

The listing of the previously mentioned computer program is shown in Appendix A and sample inputs and outputs for the program are shown in Appendix B.

The results of the two methods of fault-detection test generation are applicable only to the steady-state behavior of combinational circuits. Within the realm of combinational circuits, single-output, multiple-level circuits are considered in this thesis. Figure 4 shows a typical multiple-level combinational circuit with single output.

The type of faults studied in this research are logical faults; faults which produce some change in the logical behavior of the circuit. The malfunctioning of a circuit may be due to a permanent fault or an intermittent one. Intermittent faults such as hazards are faults which may appear, disappear or change their nature during testing and will not be considered in this research. The prime consideration of this research is faults of the permanent (solid)

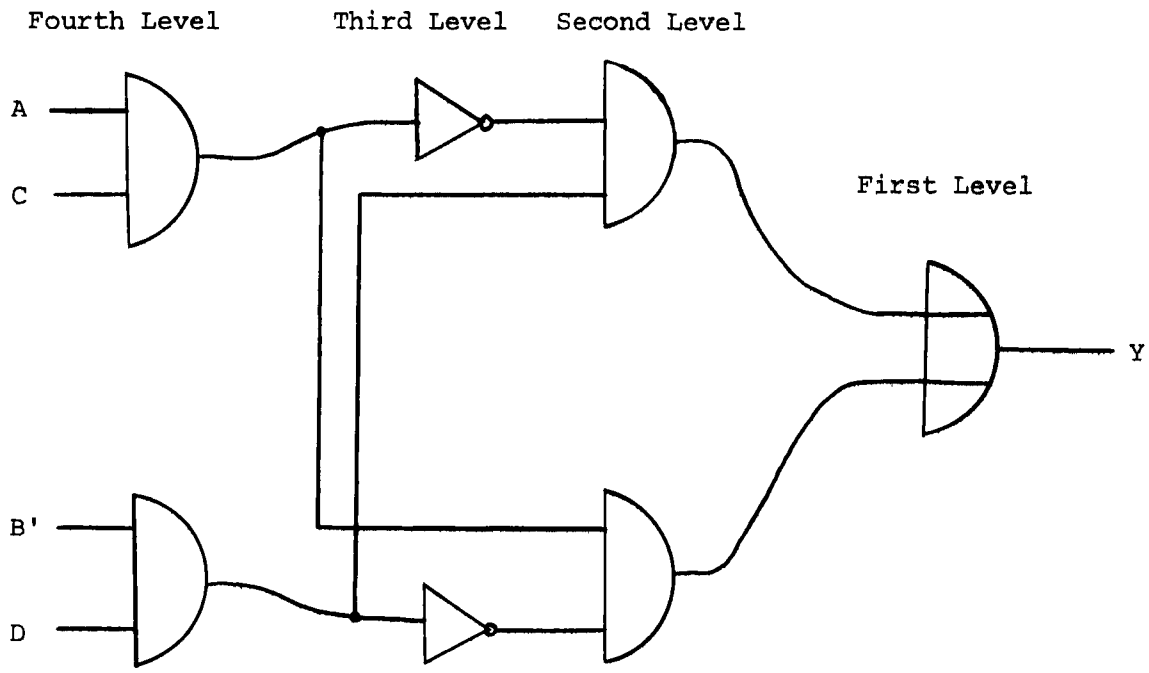


Figure 4. Typical single-output, multiple-level circuit.

type which are caused by hardware component failure (2). These faults are studied under the assumption that the circuit cannot have more than one fault at any time.

CHAPTER II

FAULTS IN COMBINATIONAL CIRCUITS

The logical behavior of a combinational function is affected when a gate malfunctions causing the transformation of the combinational function into a different combinational function. This research is only concerned with the effect that a gate failure has upon this behavior and not with the cause of the failure. The cause of gate failures is dependent upon the type of logic circuit used to realize the combinational function. The effect of a gate failure is that one of the gate inputs or the gate output is either permanently stuck at a logical value of "1" or "0". These permanent logical faults denoted as stuck-at-one (s-a-1) and stuck-at-zero (s-a-0) are independent of the inputs applied to the circuit (1).

A stuck-at-zero fault occurs when a wire in the normal circuit becomes stuck at the logical value of "0". The behavior of a gate when one of the gate inputs becomes s-a-0 depends upon the type of gate and the other inputs to the gate. A s-a-0 input to an AND gate will always cause the gate output to be 0 while a s-a-0 input to an OR gate will cause the gate output to be 0 if all the other inputs to the gate are 0, otherwise the gate output will be 1.

Figure 5 illustrates the gate behavior for the gate types used in this research when one of the gate inputs is s-a-0.

Stuck-at-one faults occur much the same as s-a-0 with the exception that they occur when a wire in the normal circuit becomes stuck at the logical value of "1". The effect that s-a-1 faults have upon the gate output is also dependent upon the gate type and the other inputs to the gate. The gate behavior that a s-a-1 fault at a gate input has upon the gate output is shown in Figure 6.

Stuck-at-zero and stuck-at-one faults can also occur at the output of a gate. A gate output that is s-a-0 or s-a-1 and that is the input to one or more other gates will cause these inputs to also be faulty. This fault may even propagate to the output of the circuit and cause the circuit output to be different from its normal value. A s-a-0 or s-a-1 fault that propagates to the circuit output is said to be detectable. Figure 7b is an example of a faulty circuit where the fault x s-a-1 propagates to the circuit output while Figure 7a shows the normal circuit. In Figure 7b the behavior of the circuit is illustrated when the AND gate fails at the input wire labeled x causing x to become s-a-1. Since the other input to the AND gate is 1 for this input combination, the gate output becomes 1. Thus, the input to the OR gate becomes 1 which causes the output of the OR gate to become 1. With the normal circuit output being 0 and the

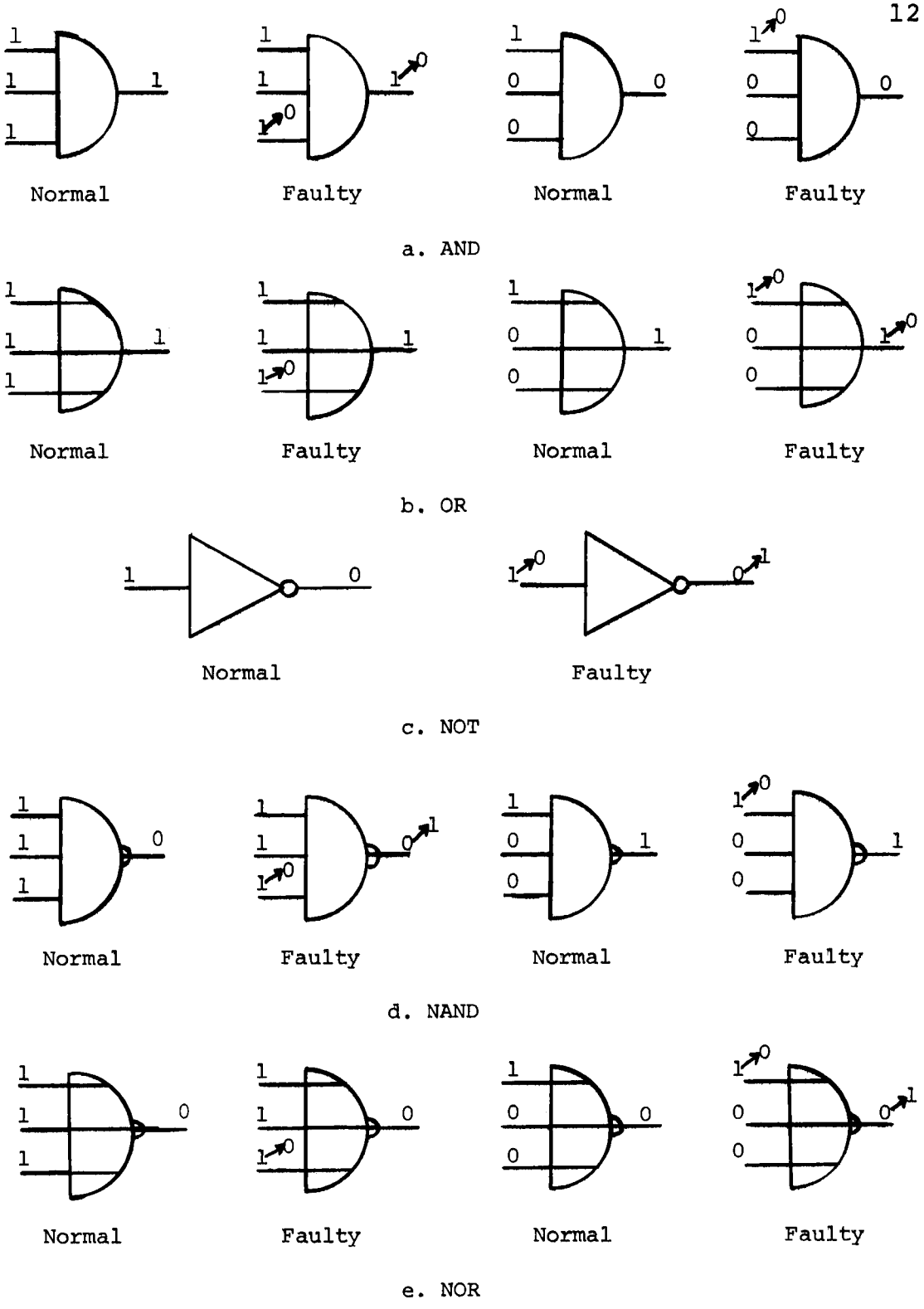


Figure 5. Gate behavior when a gate input is s-a-0.

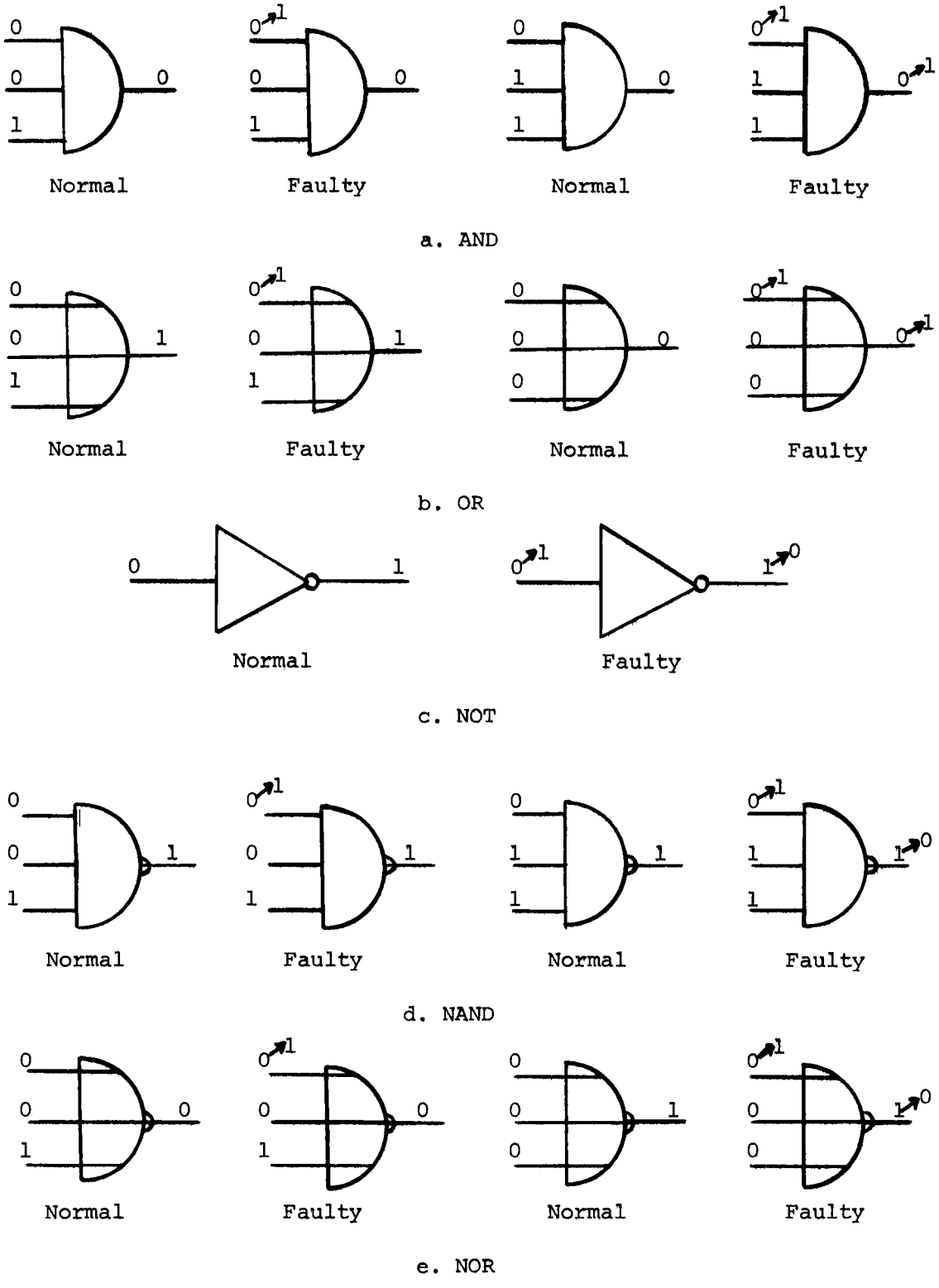
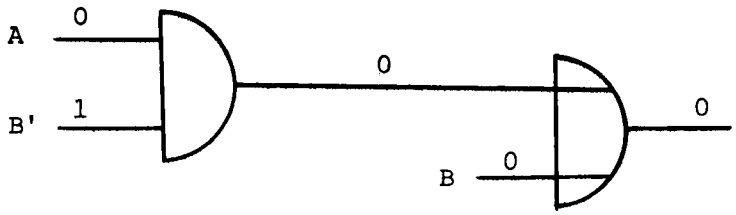
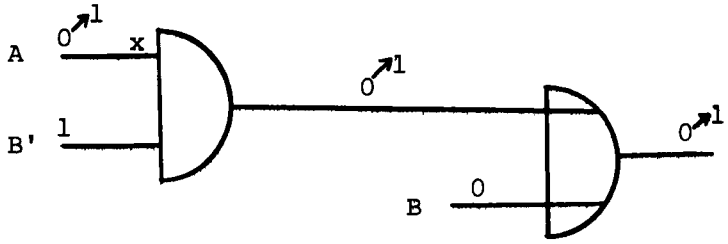


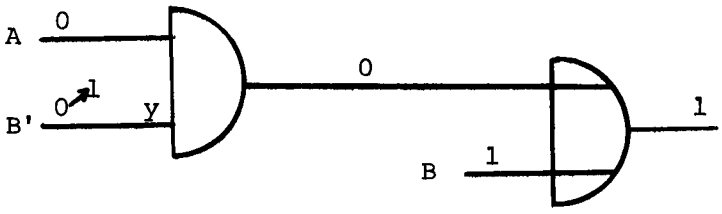
Figure 6. Gate behavior when a gate input is s-a-1.



a. Normal circuit for input combination A = 0, B = 0.



b. Faulty circuit with detectable fault
x s-a-1 for input combination A = 0, B = 0.



c. Faulty circuit with undetectable fault
y s-a-1 for input combination A = 0, B = 1

Figure 7. Normal and faulty circuit behavior.

circuit output being 1 when the wire x is s-a-1, the wire x is detectable for the fault s-a-1.

An undetectable fault is a fault which does not cause a change in the normal output of the circuit for all possible input combinations applied to the circuit. The AND gate has failed in Figure 7c causing the input wire labeled y to become s-a-1. With the other input to the AND gate being 0, the output from the AND gate is 0 regardless of the value of y. Since y s-a-1 has no effect on the gate output, and therefore, can't propagate to the circuit output, the wire y s-a-1 is undetectable for the input combination considered. The fact that y s-a-1 is undetectable for this input combination does not imply that this fault is not detectable for some other input combination. One would have to apply all the possible input combinations before he could state that y s-a-1 was undetectable. In this circuit, when all the possible input combinations are applied, y s-a-1 is undetectable.

From looking at the sum of products form of the circuit in Figure 7, AB' + B, one can immediately notice that the circuit is redundant and can be reduced to A + B. This leads to the conclusion that undetectable faults and redundant circuits are somewhat related. In fact, the definition of redundancy may be given in terms of undetectable faults: A circuit is redundant if and only if it contains some fault

of the line stuck type which is undetectable (5). Redundancy is undesirable in combinational circuits from the point of view of testing, because some faults in a redundant circuit are undetectable. The presence of an undetectable fault may cause some other detectable fault to become undetectable by a complete set of tests derived under the single-fault assumption (2).

Undetectable faults and irredundant circuits are also related. A circuit is said to be irredundant if and only if no input lines or internal lines can be s-a-1 or s-a-0 without changing the output and will be detectable (6). All circuits studied in this research are irredundant and thus every single failure will be detectable.

CHAPTER III

FAULT-DETECTION TEST GENERATION

A fault-detection test refers to a single application of values to the input terminals. The simplest way of constructing fault-detection tests involves the use of fault tables. A fault table is a table in which there is a row for every possible test (i.e., input combination) and a column for every fault. Two methods of generating fault-detection tests and thus, constructing the fault table are presented in this chapter.

I. TABLE OF COMBINATIONS METHOD

The table of combinations method for generating fault-detection tests is only valid under the single-fault assumption for irredundant circuits. This limitation does not, of course, exclude the detection of most double and other multiple faults, but the limitation emphasizes that only single faults will be detected in all cases, while some multiple faults may not be detected. From a practical point of view this is not a severe limitation because the probability of occurrence of multiple faults is rather small in comparison to the probability of a single fault occurring (1).

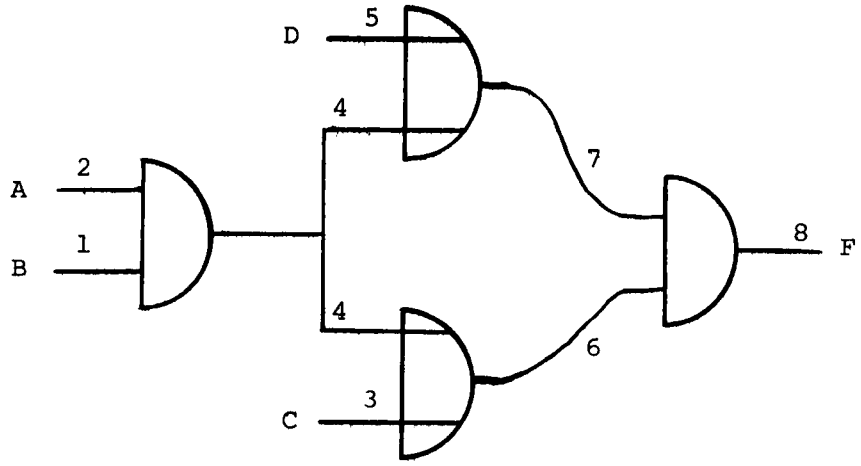
By this method the fault table may be constructed one row at a time or one column at a time. In the first approach

all faults detected by a given input combination are formed while in the second approach all possible tests detecting a given fault are determined. The fault tables studied in this research are constructed one row at a time.

Since any wire in a combinational circuit may be either s-a-0 or s-a-1, the fault table for a circuit with k input and internal wires or nodes would have 2k columns or 2k possible faults. The columns in the fault table are labeled to correspond to the type of fault and location of the fault within the circuit. Figure 8 shows an example of how the columns of the fault table correspond to the wires of the circuit. Each wire in the circuit is given a unique label with the exception of a gate output wire which has a fanout greater than 1. Gate inputs which are driven by the same gate output are given identical labels since a single fault on any of these wires will also affect the others. This is shown in Figure 8a where the output from gate 1 is the input to both gate 2 and gate 3.

Fault-detection tests can be generated by the table of combinations method with either of two approaches. The first approach checks each wire of the circuit for either s-a-0 or s-a-1 for each possible input combination. This approach is applied using the following steps.

Step 1. Apply input combination i to generate the normal value of each wire and the normal circuit output, F.



a. Circuit with labeled wires.

INPUTS	CORRECT OUTPUT	POSSIBLE S-A-0 FAULTS								POSSIBLE S-A-1 FAULTS							
		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
ABCD	F	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0000	0												X				X
0001	0											X	X		X		X
0010	0												X	X		X	X
0011	1			X		X	X	X	X								
0100	0										X		X				
0101	0										X	X	X		X		X
0110	0										X		X	X		X	X
0111	1			X		X	X	X	X								
1000	0									X			X				X
1001	0									X		X	X		X		X
1010	0									X			X	X		X	X
1011	1			X		X	X	X	X								
1100	1	X	X		X		X	X	X								
1101	1	X	X		X		X	X	X								
1110	1	X	X		X		X	X	X								
1111	1						X	X	X								

b. Fault table for above circuit.

Figure 8. A circuit and its fault table generated by table of combinations method.

Step 2. Assign to wire j its opposite value and generate the trial circuit output, F'.

Step 3. If F = F', go to step 5.

Step 4. A fault at wire j can be detected by input combination i. If wire j is 0, place an X at the intersection of the ith row and the jth column in the s-a-0 section of the fault table, else place an X at the intersection of the ith row and the jth column in the s-a-1 section of the fault table.

Step 5. Restore wire j to its original value and repeat steps 2 thru 5 until all wires in the circuit have been checked for input combination i.

Step 6. Repeat steps 1 thru 5. Procedure terminates when all possible input combinations have been applied.

Consider the circuit in Figure 8a. When input combination 0000 is applied to the circuit, all wires in the circuit have a logical value of 0 and the normal circuit output is 0. Since wire 1 is 0, step 2 assigns wire 1 the value of 1 and generates the trial output of 0. The normal output and the trial output when wire 1 is s-a-1 are equal, therefore, step 4 is omitted. Step 5 restores wire 1 to 0 and continues with step 2 until all wires in the circuit have been checked for input combination 0000. Repeated applications of this approach will generate the complete fault table as shown in Figure 8b.

While the first approach checks each wire in the circuit beginning with wire 1 and proceeding to the circuit output, the second approach begins with the circuit output and proceeds backwards through the circuit only considering faulty wires which can be detected. This approach is faster than the first, because only the wires which can be detected faulty are checked. The fault table can be constructed with this approach by applying the following steps.

Step 1. Apply input combination i to generate the normal value of each wire and the normal circuit output, F.

Step 2. If wire j corresponding to the circuit output is 0, place an X at the intersection of the ith row and jth column in the s-a-1 section of the table, else place an X at the intersection of the ith row and jth column in the s-a-0 section of the table. Each input to the output gate represents a path back through the circuit on which faults may be detected. These paths may branch many times depending on the gate types.

Step 3. If gate k (k is the level of gating) is an AND or NAND gate and if 2 or more of the inputs to gate k are 0 or if gate k is an OR or NOR gate and if 2 or more of the inputs to gate k are 1, go to step 6. If gate k is an AND or NAND gate and its output is 0 or if gate k is an OR or NOR gate and its output is 1, go to step 5.

Step 4. All inputs to gate k represent a path on which faults may be detected. If input m, wire j, to gate k is 1, place an X at the intersection of the ith row and the jth column in the s-a-0 section of the table, else place an X at the intersection of the ith row and the jth column in the s-a-1 section of the table. If input m is an output of another gate, gate k becomes gate k + 1 in the next higher level of gating that drives input m, go to step 3. When the last input to gate k is checked, gate k becomes gate k - 1 which the output of gate k drives in the next lower level of gating, go to step 3. Procedure terminates for input combination i when all paths from the inputs to the output gate have been checked, go to step 7.

Step 5. Only one path from the inputs to gate k can be checked. If this path has already been checked, gate k becomes gate k - 1 which the output of gate k drives in the next lower level of gating, go to step 3. If gate k is an AND or NAND gate and input wire j is 0, place an X at the intersection of the ith row and jth column in the s-a-1 section of the table. If gate k is an OR or NOR gate and input wire j is 1, place an X at the intersection of the ith row and jth column in the s-a-0 section of the table. Gate k becomes the gate k + 1 that drives input wire j in the next higher level of gating, go to step 3. Procedure terminates for input combination i when all paths from the inputs to the output gate have been checked, go to step 7.

Step 6. None of the inputs to gate k can be detected faulty. Gate k becomes gate k - 1 which drives the output of gate k in the next lower level of gating. Procedure terminates for input combination i when gate k is the output gate, go to step 7.

Step 7. Steps 2 thru 6 will detect all single faults except some gate output wires which have a fanout greater than 1 for input combination i. Assign to the output, wire j, of gate m (m is a gate with fanout > 1) its opposite value and generate a trial circuit output, F'. If F = F', go to step 8, else go to step 9.

Step 8. Repeat step 7 until all gates with fanout > 1 have been checked, go to step 10.

Step 9. If the output of gate m is 0, place an X at the intersection of the ith row and the jth column in the section of the fault table, else place an X at the intersection of the ith row and jth column in the s-a-1 section of the fault table. Consider gate m as the circuit output gate and go to step 2.

Step 10. Go to step 1 and apply next input combination. Procedure terminates when all possible input combinations have been applied.

Consider the circuit in Figure 8a, page 19. When input combination 0000 is applied to the circuit, all wires in the circuit have a value of 0 and the normal circuit output is 0.

By step 2 of this approach, an X is placed in column 8 of the s-a-1 section of the table. Since the output gate is an AND gate with all its inputs 0, steps 4 and 5 are skipped. Step 6 transfers execution to step 7, where all gates with fanout greater than 1 are checked. Step 7 places an X in column 4 of the s-a-1 section of the fault table. Repeated applications of this approach will also generate the fault table as shown in Figure 8b, page 19.

II. POAGE'S METHOD

Poage's method can be used to generate fault-detection tests for both single and multiple faults occurring in a combinational circuit (7). This method uses an analysis procedure which yields an output expression incorporating circuit structure in addition to input variables. This expression or propagation represents not only the relationship of the gate outputs to the input and internal lines, but also the effect of faults on the gate outputs.

Each wire in the circuit can be in one of three states, normal, s-a-0 or s-a-1. Three binary variables are used to specify the state of each wire x_i in the circuit and are defined as follows:

x_{in} = 1 if and only if x_i is normal

x_{i1} = 1 if and only if x_i is s-a-1

x_{i0} = 1 if and only if x_i is s-a-0

For each wire x the totality of conditions under which wire x can have a value of 1 is represented by the proposition P_x and the conditions under which wire x can have a value of 0 is represented by P'_x. Consider the circuit in Figure 9a. If B is the input for wire a, then the proposition for wire a having a value of 1 is $P_a = Ba_n + a_1$ which means wire a will have a value of 1 if B = 1 and wire a is normal or if wire a is s-a-1. For the circuit in Figure 9a, the output propositions P_g and P'_g are obtained as shown below:

$$P_a = Ba_n + a_1$$

$$P'_a = B'a_n + a_0$$

$$P_b = A'b_n + b_1$$

$$P'_b = Ab_n + b_0$$

$$P_c = B'c_n + c_1$$

$$P'_c = Bc_n + c_0$$

$$P_d = Ad_n + d_1$$

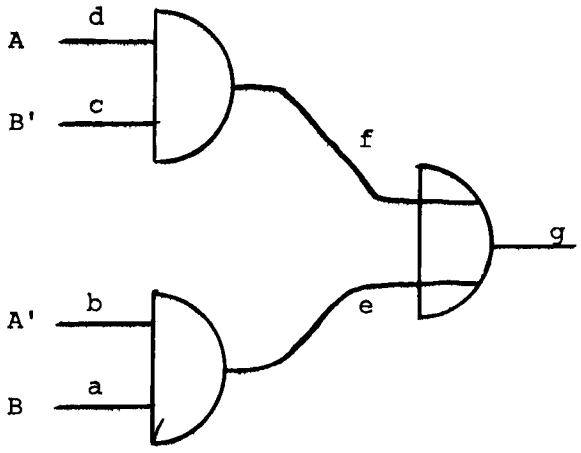
$$P'_d = A'd_n + d_0$$

$$P_e = (P_aP_b)e_n + e_1$$

$$= (Ba_n + a_1)(A'b_n + b_1)e_n + e_1$$

$$P'_e = (P'_a + P'_b)e_n + e_0$$

$$= (B'a_n + a_0 + Ab_n + b_0)e_n + e_0$$



a. Circuit for AB' + A'B.

INPUTS	POSSIBLE S-A-0 FAULTS							POSSIBLE S-A-1 FAULTS						
	a	b	c	d	e	f	g	a	b	c	d	e	f	g
AB	0	0	0	0	0	0	0	1	1	1	1	1	1	1
00								X			X	X	X	X
01	X	X			X		X							
10			X	X		X	X							
11									X	X		X	X	X

b. Fault table for above circuit.

Figure 9. A circuit and its fault table generated by Poage's method.

$$P_f = (P_c P_d) f_n + f_1$$

$$= (B'c_n + c_1)(Ac_n + c_0) f_n + f_1$$

$$P'_f = (P'_c + P'_d) f_n + f_0$$

$$= (Bc_n + c_0 + A'd_n + d_0) f_n + f_0$$

$$P_g = (P_e + P_f) g_n + g_1$$

$$= [(Ba_n + a_1)(A'b_n + b_1) e_n + e_1 + (B'c_n + c_1)$$

$$(Ac_n + c_0) f_n + f_1]\ g_n + g_0$$

$$P'_g = (P'_e P'_f) g_n + g_0$$

$$= [(B'a_n + a_0 + Ab_n + b_0) e_n + e_0][(Bc_n + c_0 + A'd_n$$

$$+ d_0) f_n + f_0] g_n + g_0$$

The effect of any fault on the output can be determined by assigning appropriate values to the element variables (wires). For example, if wire a is s-a-1, set $a_n = a_0 = 0$, $a_1 = 1$, all other element variables with subscript n at 1 and the remaining element variables at 0. Denoting the output propositions of the normal circuit by $P_g(n)$ and $P'_g(n)$ and those of the faulty circuit by $P_g(a_1)$ and $P'_g(a_1)$, the following expressions are derived:

$$P_g(n) = BA' + B'A$$

$$P'_g(n) = (B' + A)(B + A')$$

$$P_g(a_1) = A' + B'A$$

$$P'_g(a_1) = ABA' = 0$$

The set of tests for detecting a s-a-1 fault on wire a is given by:

$$\begin{aligned} T(a_1) &= P_g(a) \cdot P'_g(a_1) + P'_g(n) \cdot P_g(a_1) \\ &= (BA' + B'a)0 + (B' + A)(B + A')(A' + B'A) \\ &= 0 + (AB + A'B')(A' + B'A) \\ &= A'B' \end{aligned}$$

The input combination 00 is the only test for wire a s-a-1. Place an X at the intersection of row 00 and column a in the s-a-1 section of the fault table. Each input wire and internal node in the circuit is checked for s-a-0 and s-a-1 and thus completing the fault table one column at a time as shown in Figure 9b, page 26.

Poage's method can also be used to detect multiple faults. In spite of this advantage, this method is only practical for relatively small circuits because of the complexity of the output propositions of larger circuits (2,7).

CHAPTER IV

THE FAULT TABLE

Whether derived by the table of combinations method or by Poage's method, the fault table represents all single stuck-at-one or stuck-at-zero faults which may occur within a combinational circuit. The table, which has a row for each test and a column for each fault, is partioned into two sections—"possible s-a-0 faults" and "possible s-a-1 faults". Each of these sections contains a column for each input wire and internal node in the circuit. In the table every entry c_{ij} has an X if row i detects a fault at column j and blank otherwise (1).

Minimal sets of tests for fault detection are obtained by finding the smallest set of rows such that every column of the appropriate fault table has an X in at least one of the rows in the set. Finding a minimal set of tests is similar to the prime implicant covering problem of McCluskey (8). The prime implicant covering problem removes essential rows in the minimal solution, and removes certain other rows and columns by row and column dominance techniques. Methods used to solve the prime implicant covering problem will be used in this research to cover a fault table, and thus find a minimal fault-detection experiment.

I. ESSENTIAL TESTS

An essential test is a test that must be included in the minimal fault-detection experiment. Essential tests can be either primary or secondary. A primary essential test is the only test that will detect a given fault, while a secondary essential test is one that is not essential until dominating rows are removed from the table. For a column containing a single X, the row in which the X occurs corresponds to an essential test. An example of a primary essential test is row 01 in the fault table in Figure 9, page 26. The columns of the fault table are scanned left to right. When an essential test is noted, row i is added to the minimal experiment and all columns of the fault table with an X in row i are removed from the fault table.

II. ROW DOMINANCE

A fault table can also be reduced by removing dominated rows. A row i is said to dominate row j if row i has an X in every column in which row j has an X (1,8). Shown below is an example of row dominance:

A	X	X	X
B		X	X

Row B is dominated by or contained in row A and can be eliminated because any fault detected by row B will also be detected by row A. For this reason any dominated row is removed from the fault table. Equal rows present a special case of row dominance. Two rows are equal if each row has an X in every column that the other has. For equal rows, an arbitrary choice is made and either of the rows can be removed from the fault table.

III. COLUMN DOMINANCE

Dominating columns can also be used to reduce a fault table. A column i is said to dominate column j if column i has an X in every row in which column j has an X (1,8). The dominating column i is removed, since any test which detects fault j will also detect fault i. In the example below column c dominates column b and can be eliminated.

	b	c
A	X	X
B		X
C	X	X

The elimination of column c is justifiable because any test which detects the fault represented by column b also detects the fault represented by column c. Equal columns are a special case of column dominance. Two columns are equal if

each column has an X in every row that the other has. An arbitrary choice is made to remove either of the equal columns from the fault table.

IV. CYCLIC TABLE

A cyclic table is a table which has no essential tests, dominated rows, or dominating columns (1,8). Cyclic tables usually occur during the process of minimizing the original fault table. An example of a cyclic table is shown in Figure 10a. Whenever a cyclic table is encountered, an approach called the branching method is used to complete the minimization of the fault table (1,3,8).

To find a minimal cover for the cyclic table, one should make an arbitrary selection of one row and then apply the techniques for essential tests, dominated rows and dominating columns to minimize the remaining table. For example in the cyclic table in Figure 10a, column 1 can be covered by either row 000 or row 111; consequently one of these rows must be in the minimal cover. If row 000 is arbitrarily chosen, the table in Figure 10b results. This table can now be reduced by the previously mentioned techniques, and a minimal cover containing row 000 is obtained for the cyclic table. The entire process must now be repeated with row 111 arbitrarily chosen to cover column 1 as shown in Figure 10c. The minimal cover for the cyclic table of Figure 10a is

	1	3	7	9	10
000	X	X			
001		X	X		
100			X	X	
110		X		X	X
111	X		X		

a. Cyclic table.

	7	9	10
001	X		
100	X	X	
110		X	X
111	X		X

b. Reduced table after selection of row 000.

	3	9
000	X	
001	X	
100		X
110	X	X

c. Reduced table after selection of row 111.

Figure 10. Minimization by the branching method.

obtained by choosing the smaller of the two minimal solutions which cover column 1.

For large cyclic tables, this method could have many branches. Moreover, a cyclic table may result while applying the branching process and reducing another cyclic table. To avoid the possibility of a cyclic table having many branches, a method of combinations is used to reduce cyclic tables encountered in this research. For a cyclic table with n rows, the combinations of n rows taken r at a time are examined until a minimal cover is found for the table. Beginning with r = 2 and continuing until a minimal cover is obtained or until r = n insures that the cover obtained is minimal.

The minimal cover for a cyclic fault table, whether obtained by the branching method or the method of combinations, is added to the essential tests obtained by removing dominated rows and dominating columns to give a minimal fault-detection experiment.

V. MINIMAL FAULT-DETECTION EXPERIMENT

The techniques discussed previously in this chapter can be applied in a stepwise procedure to derive a minimal set of fault detection tests. The steps in this procedure are:

Step 1. Scan the columns of the fault table left to right until all essential tests have been removed from the fault table and added to the minimal set of tests.

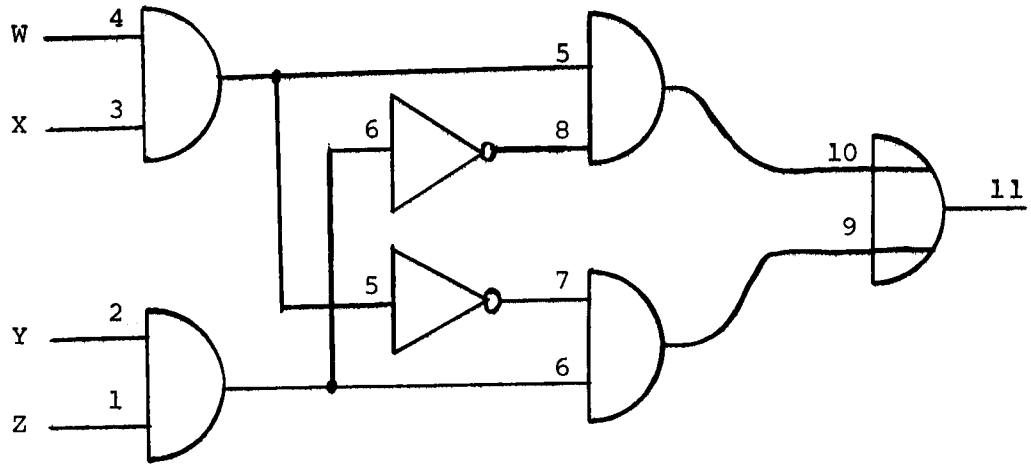
Step 2. Remove all dominated rows and equal rows from the fault table. If at least one row was removed, go to step 1 and check for secondary essential tests, else go to step 3.

Step 3. Remove all dominated columns and equal columns from the fault table. If at least one column was removed, go to step 2, else go to setp 4.

Step 4. If the fault table doesn't have any columns remaining, a minimal cover has been obtained and the procedure terminates, else go to step 5.

Step 5. The fault table is cyclic, apply the combinations of n rows taken r at a time and beginning with r = 2 until a minimal cover for the cyclic table is obtained. Add this minimal cover to the minimal set of tests obtained above.

Consider the circuit and fault table in Figure 11. Step 1 of the procedure detects the primary essential test 1111. Row 1111 is added to the minimal set of fault-detection tests and is removed from the fault table as shown in Figure 12a. Since no other primary essential tests are present, step 2 removes all dominated rows. The next step is to apply step 1 and scan the reduced fault table for secondary essential tests. With no secondary essential tests and no dominated rows present, step 3 is used to remove the dominating columns. Applying steps 1, 2 and 3



a. Circuit.

INPUTS	CORRECT	POSSIBLE S-A-0 FAULTS											POSSIBLE S-A-1 FAULTS										
	OUTPUT	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9	10	11
WXYZ	F	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0000	0																X	X			X	X	X
0001	0													X			X	X			X	X	X
0010	0												X				X	X			X	X	X
0011	1	X	X				X	X		X		X					X						
0100	0															X	X	X			X	X	X
0101	0													X		X	X	X			X	X	X
0110	0												X			X	X	X			X	X	X
0111	1	X	X				X	X		X		X				X	X						
1000	0														X		X	X			X	X	X
1001	0													X	X		X	X			X	X	X
1010	0												X		X		X	X			X	X	X
1011	1	X	X				X	X		X		X			X		X						
1100	1			X	X	X			X		X	X						X					
1101	1			X	X	X			X		X	X		X				X					
1110	1			X	X	X			X		X	X	X					X					
1111	0	X	X	X	X	X	X												X	X	X	X	X

b. Fault table.

Figure 11. Circuit and its fault table.

INPUTS	CORRECT OUTPUT	POSSIBLE S-A-0 FAULTS					POSSIBLE S-A-1 FAULTS					
		7	8	9	10	11	1	2	3	4	5	6
WXYZ	F	0	0	0	0	0	1	1	1	1	1	1
0000	0										X	X
0001	0							X			X	X
0010	0						X				X	X
0011	1	X		X		X					X	
0100	0									X	X	X
0101	0							X		X	X	X
0110	0						X			X	X	X
0111	1	X		X		X				X	X	
1000	0								X		X	X
1001	0							X	X		X	X
1010	0						X		X		X	X
1011	1	X		X		X			X		X	
1100	1		X		X	X						X
1101	1		X		X	X		X				X
1110	1		X		X	X	X					X

a. Reduced table after removal of essential test, row 1111.

INPUTS	CORRECT OUTPUT	POSSIBLE S-A-0 FAULTS		POSSIBLE S-A-1 FAULTS		
		7	8	1	2	3
WXYZ	F	0	0	1	1	1
0101	0				X	X
0110	0			X		X
0111	1	X				X
1001	0				X	X
1010	0			X		X
1011	1	X				X
1101	1		X		X	
1110	1		X	X		

b. Cyclic table.

Figure 12. Minimization of fault table.

yield no further reductions; therefore, the remainder of the fault table is cyclic as shown in Figure 12b. The cyclic table is reduced by step 5 and the minimal fault-detection experiment shown below is derived.

1. Apply 1111: If the output is F = 1, circuit is faulty and the experiment may be stopped; otherwise, apply next test.
2. Apply 0111: If the output is F = 0, circuit is faulty and the experiment may be stopped; otherwise, apply next test.
3. Apply 1010: If the output is F = 1, circuit is faulty and the experiment may be stopped; otherwise, apply next test.
4. Apply 1101: If the output is F = 1, circuit is faulty; otherwise, circuit operates correctly.

CHAPTER V

MINIMIZATION WITH THE AID OF A DIGITAL COMPUTER

The digital computer program described in this chapter is designed to accept the realization of a combinational circuit as input, generate the fault table and find a minimal fault-detection experiment for the circuit. This program is written in FORTRAN IV and ALC for use with the IBM/360 MODEL 65 computer. Since the theoretical basis of the program is the second approach to the table of combinations method as outlined in Chapter III, no further elaboration will be given here concerning this aspect of the problem. This chapter will explain the program's input sequence and techniques used in writing the program. The program listing is shown in Appendix A and sample inputs and outputs are shown in Appendix B.

Several general observations of the program's purpose and limitations should be pointed out before explaining the specific input sequences. The program will accept a minimum of one gate and a maximum of one hundred gates; the minimum number of input variables is two and the maximum is seven. For any individual gate the maximum number of gate inputs is eight and the maximum fanout is ten. The program is

designed to handle single faults in irredundant, single-output combinational circuits.

The input card sequence used with the program is:

Card 1. Number of input variables and gates.

Columns 1-3. Number of gates (right justified).

Columns 4-6. Number of input variables (right justified).

Card 2. Input variables and complements. Each variable and complement is right justified in a three column field. The first field begins in column 1. All input variables are punched in the card followed by their complements in the same order.

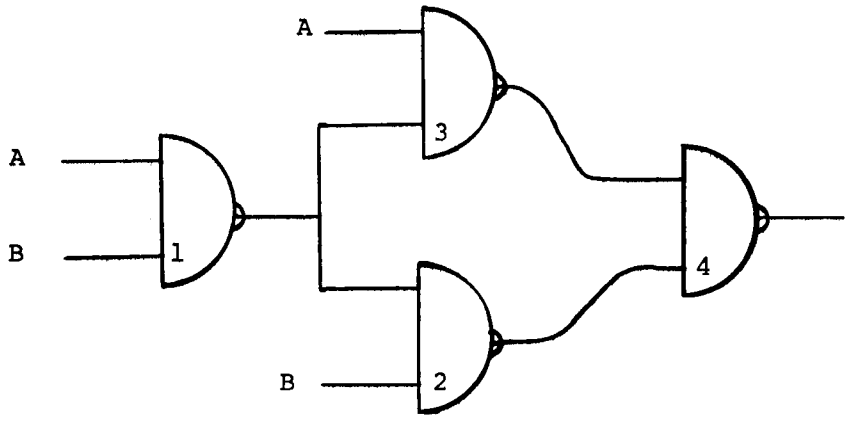
Card 3 up to 102. Individual gates. The gates are numbered as shown in Figure 13. All the first level gates are numbered first, followed by the next level and so on. Gate 1 is placed on card 3, gate 2 on card 4, etc. The last card must be the output gate.

Columns 1-4. Gate type (left justified).

Columns 7-8. Number of gate inputs (right justified).

Columns 11-12. Fanout of gate (right justified).

Columns 13-16 up to 41-44. Gate inputs (right justified).



a. Circuit for AB' + A'B.

```
Card 1.   4  2
Card 2.     A  B  A' B'
Card 3.  NAND   2  2  B  A
Card 4.  NAND   2  1  B  1
Card 5.  NAND   2  1  1  A
Card 6.  NAND   2  1  2  3
```

b. Input card sequence for program.

Figure 13. A circuit and its input data.

Figure 13 illustrates the use of the previously mentioned input scheme. Card 1 contains the number of gates, 4, and the number of input variables, 2. The second input card contains the input variables, AB, and their complements, A'B'. Since the circuit is realized with four gates, there are four cards which follow card 2. Each of these cards contains the gate type, number of gate inputs, fanout of gate and gate inputs. A gate input represented on a card is either an input variable, a complement of an input variable or the gate number which drives the input.

The manner in which the circuit is represented on the input cards determines how the program will generate the faults and construct the fault table. For example, Figure 14 illustrates how different numbering of gates within one level will cause faults to be labeled differently. Also, the order in which a gate's inputs are placed on the card will affect the labeling of faults. The gate inputs are numbered in the order in which they appear on the card. Even though the order of gate inputs and the numbering of gates within a level affect the labeling of faults, they have no effect on the number of tests in the minimal fault-detection experiment.

The program output consists of a print of the input data, the fault table and the minimal fault-detection experiment. The format of the fault table follows the pattern of the thesis text. Information concerning the nature of the

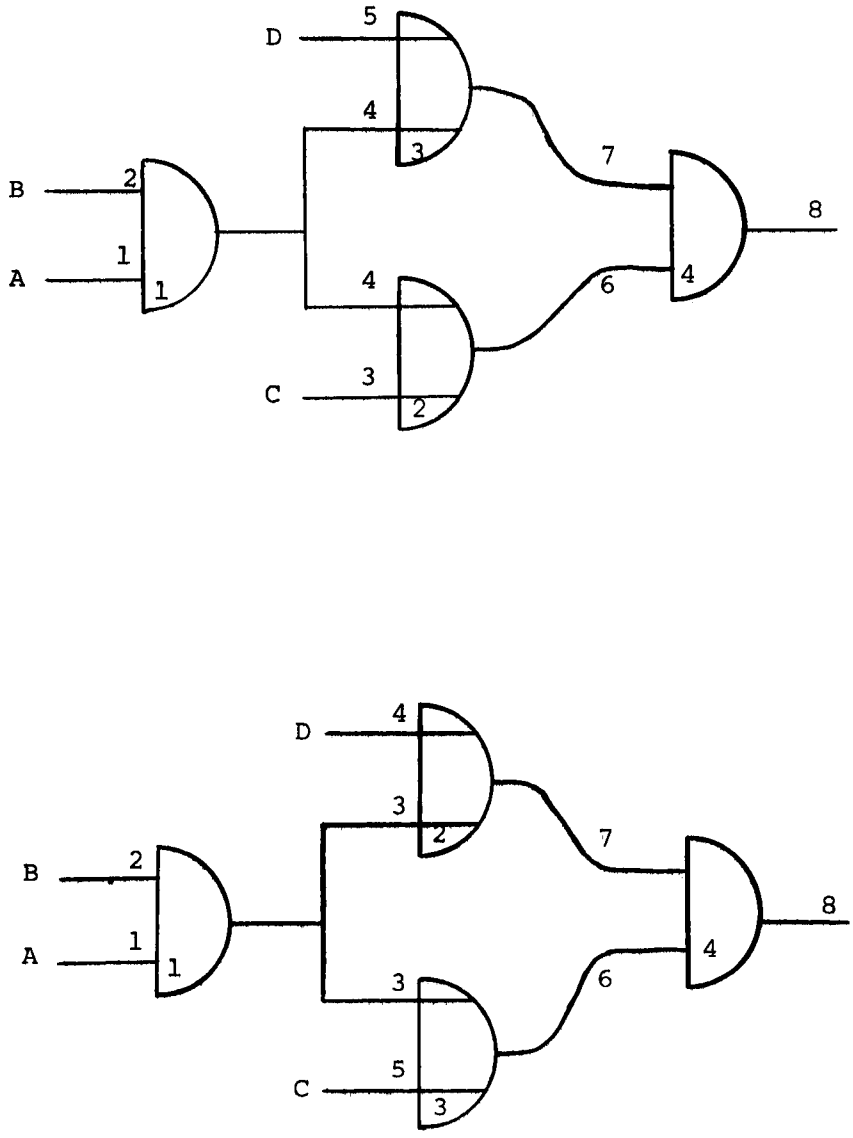


Figure 14. Two examples of different numbering of gates in a circuit.

minimal fault-detection experiment is printed and should be self-explanatory.

The program consists of three major divisions; generating the fault table, reducing the fault table and printing the results. Since each of these divisions can be executed independently of each other, only one division needs to be in the computer's main memory at one time. This can be accomplished by a process called overlay. Limitations in the amount of available memory space caused this overlay process to be instituted. Also, one and two byte words were used to overcome this limitation.

A short main routine resides in memory and controls the execution of the program by initiating the overlay process at the proper points. The main routine and one of the program's major divisions remains in the computer's main memory throughout the execution of the program. Each division is composed of several subroutines to handle different repetitive tasks.

The division of the program for generating the fault table is composed of subroutines INP, CONVT, IGATE, NGATE, TABLE, and FANOUT. Subroutine INP is the controlling subroutine for this division. The input data is read in, validated and printed. An ALC subroutine, CONVT, is used to validate the input data. The data is checked for gate type, number of gate inputs and gate fanout. Input combinations

are generated and function subprogram IGATE is used to find the circuit output. Subroutine TABLE controls the backtracking path and individual gates as they are checked for possible faults. The gate inputs are examined for possible faults by subroutine NGATE and all gates with fanout greater than one are examined by subroutine FANOUT.

The minimal fault-detection experiment is produced by another major division of the program. This division is controlled by subroutine MIN and consists of subroutines RECNT, COLUMN, REDUCE, ROWS and COMB. Subroutine RECNT counts the number of X's in each row and column of the fault table for use in removing essential tests by subroutine REDUCE, dominating columns by subroutine COLUMN and dominated rows by subroutine ROWS. If a cyclic table occurs, subroutine COMB generates the combinations which will cover the fault table. Redundant circuits are detected when a column contains no X's and the program is terminated.

The fault table and minimal experiment are printed by the last of the three major divisions of the program. Subroutines OUT and CONT make up this division. OUT is the controlling subroutine and CONT is used to print the section of a fault table which contains more than forty-four columns.

Each major division of the program and all the subroutines are explained in more detail in the program listing in Appendix A by the use of comment cards.

Variables and arrays used throughout the program are also identified. A general flowchart which illustrates the flow of the execution of the program is shown in Figure 15.

Because of the unusual nature of the program deck, a listing of all control cards and the positions of these cards relative to the entire deck are shown in Figure 16. This table shows the correct sequence when using the source deck. Figure 17 shows the correct sequence when using the object deck.

An accurate determination of running time for any given problem is difficult to estimate. The time element is dependent on several factors such as the number of input variables, the number of gates and the size of the fault table. As a general rule, one should allow one minute for each twenty-five gates in the circuit.

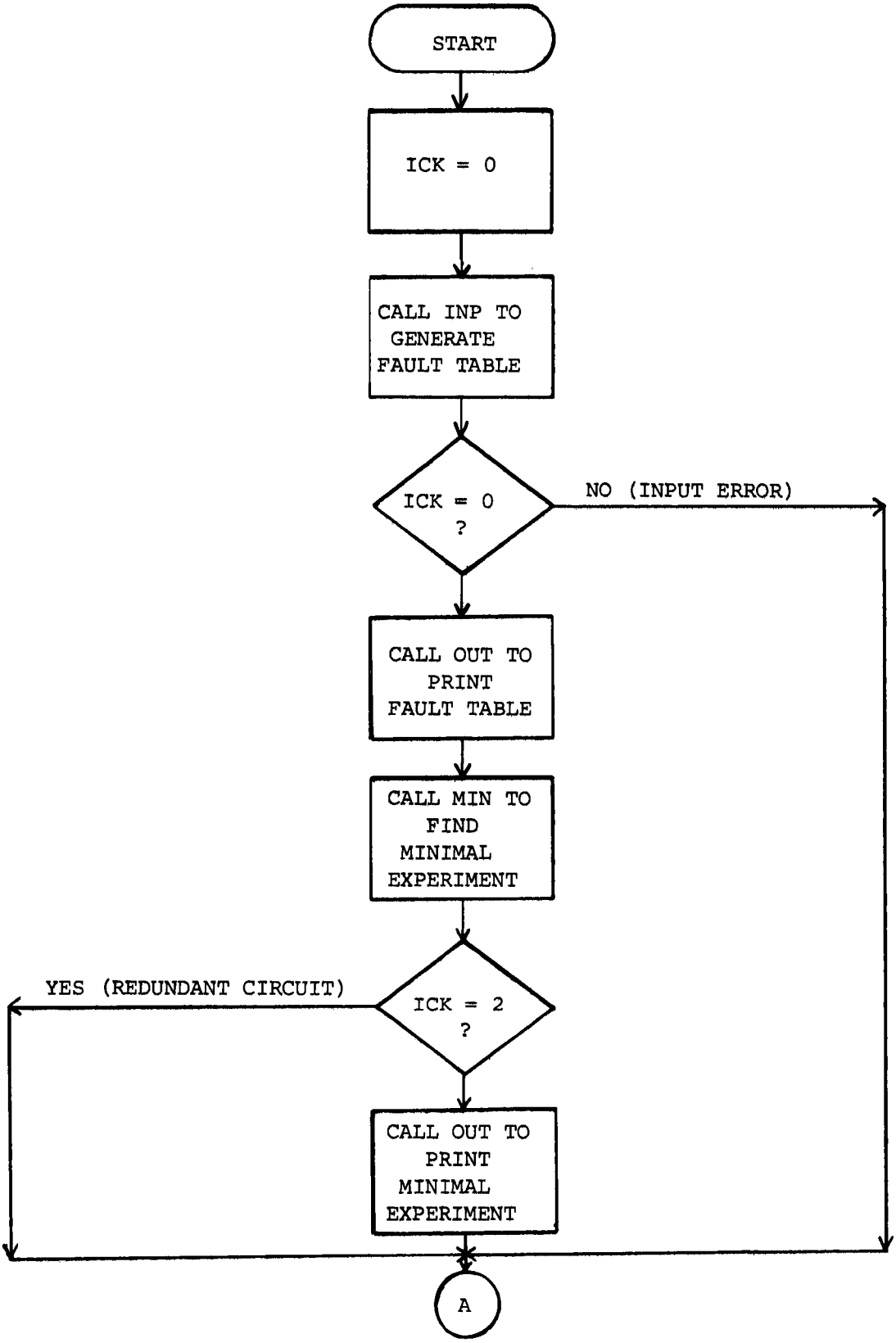


Figure 15. General flowchart for the program.

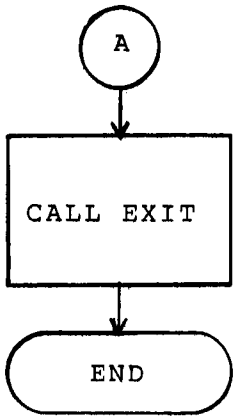


Figure 15 (continued)

```
//THESIS JOB (123456,654321,BINS,R-256K,F--66,T-1),AKIN
//STEP1 EXEC ASMGCL
//ASM.SYSIN DD *
          SUBROUTINE CONVT   (ASSEMBLER)
//STEP2 EXEC FORTGCLD,LINKFIL=MOD,LKEDOPT=OVLY,GOREG=256K
//FORT.SYSIN DD *
          MAIN PROGRAM       (FORTRAN)
          SUBROUTINE INP     (FORTRAN)
          FUNCTION IGATE     (FORTRAN)
          SUBROUTINE NGATE   (FORTRAN)
          SUBROUTINE TABLE   (FORTRAN)
          SUBROUTINE FANOUT  (FORTRAN)
          SUBROUTINE OUT     (FORTRAN)
          SUBROUTINE CONT    (FORTRAN)
          SUBROUTINE MIN     (FORTRAN)
          SUBROUTINE RECNT   (FORTRAN)
          SUBROUTINE COLUMN  (FORTRAN)
          SUBROUTINE REDUCE  (FORTRAN)
          SUBROUTINE ROWS    (FORTRAN)
          SUBROUTINE COMB    (FORTRAN)
//LKED.SYSIN DD *
          INSERT MAIN
          OVERLAY ALPHA
          INSERT INP, IGATE, NGATE, TABLE, FANOUT, CONVT
          OVERLAY ALPHA
          INSERT OUT, CONT
          OVERLAY ALPHA
          INSERT MIN, RECNT, COLUMN, REDUCE, ROWS, COMB
//GO.SYSIN DD *
          data
```

Figure 16. Control card sequence for source deck.

```
//THESIS JOB (123456,654321,BINS,R-256K,F--66,T-1),AKIN
//STEP1 EXEC FORTGLG,LKEDOPT-OVLY,GOREG=256K
//LKED.SYSIN DD *
     OBJECT DECK
          INSERT MAIN
          OVERLAY ALPHA
          INSERT INP, IGATE, NGATE, TABLE, FANOUT, CONVT
          OVERLAY ALPHA
          INSERT OUT, CONT
          OVERLAY ALPHA
          INSERT MIN, RECNT, COLUMN, REDUCE, ROWS, COMB
//GO.SYSIN DD *
     data
```

Figure 17. Control card sequence for object deck.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Present day technology requires switching circuits to be highly reliable. To meet this need for reliable circuitry, faulty circuits must first be detected and then corrected. This research has covered two methods for minimizing fault-detection experiments—table of combinations method and Poage's method.

Each method, used in conjunction with the fault table, has certain relative advantages and disadvantages and the choice of which method to use depends upon the number of faults required to be detected and, to some extent, the familiarity of the user with each method. For large numbers of input variables and gates, manual manipulation of both methods becomes very lengthy and unwieldy. For this reason, a computer program was written by the author which accepts the realization of an irredundant combinational circuit and computes the minimal fault-detection experiment to detect all single s-a-0 and s-a-1 faults which may occur in the circuit. The user of this program needs only to know how to put the data into the program and how to interpret the results.

No attempt has been made to cover any extensive segment of switching circuit reliability. The field is simply too

large for such a brief analysis. Fault-detection is only the first step in the process of maintaining a high degree of circuit reliability. The location of the fault within the circuit and either correcting or replacing the faulty component are the final steps in this process. These final steps provide areas for further research in the field of circuit reliability. Hopefully this research has served to illustrate the importance of this special area of switching theory and will lead to a better understanding of digital computer reliability.

LIST OF REFERENCES

LIST OF REFERENCES

1. Kohavi, Zvi. Switching and Finite Automata Theory. New York, 1970.

2. Friedman, Arthur D., and Premachandran R. Menon. Fault Detection in Digital Circuits. Englewood Cliffs, New Jersey, 1971.

3. Bodenheimer, Robert E. "Class Notes, EE5620," The University of Tennessee, Knoxville, 1972.

4. Chang, Herbert Y., Eric Manning and Gernot Metze. Fault Diagnosis of Digital Systems. New York, 1970.

5. Schertz, Donald R., and Gernot Metze. "On the Design of Multiple Fault Diagnosable Networks," Digest of 1971 International Symposium on Fault-Tolerant Computing, New York, 1971, pp. 46-50.

6. Russell, Jeffrey D., and C. R. Kime. "Structural Factors in the Fault Diagnosis of Combinational Networks," Digest of 1971 International Symposium on Fault-Tolerant Computing, New York, 1971, pp. 34-39.

7. Poage, J. F. "Derivation of Optimum Tests to Detect Faults in Combinational Circuits," Proceedings of Symposium on Mathematical Theory of Automata, Brooklyn, 1963, pp. 483-528.

8. McCluskey, E. J. Introduction to the Theory of Switching Circuits, New York, 1965.

APPENDIXES

APPENDIX A

LISTING OF COMPUTER PROGRAM

```
C      COMPUTER IMPLEMENTED MINIMIZATION OF FAULT-DETECTION
C      EXPERIMENTS IN COMBINATIONAL CIRCUITS
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        IN     - NUMBER OF INPUT VARIABLES(2<IN<7)         *
C*        IV     - INPUT VARIABLES                           *
C*        INPUT  - INPUT COMBINATIONS                        *
C*        IF1    - CORRECT CIRCUIT OUTPUTS                   *
C*        K      - NUMBER OF POSSIBLE FAULTS                 *
C*        K1     - NUMBER OF INPUT CONBINATIONS              *
C*        S      - FAULT TABLE                               *
C*        NUT    - INPUT COMBINATIONS IN MINIMAL EXPERIMENT  *
C*        IF2    - INCORRECT OUTPUTS IN MINIMAL EXPERIMENT   *
C*************************************************************
       COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
      1NUT(128,7),IV(14),S(128,1600)
       INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,ICK/0/
       LOGICAL*1 S
C
C      CALL SUBROUTINE INP TO INPUT DATA AND
C      GENERATE FAULT TABLE
       CALL INP(ICK)
C
C      IF ICK = 0, AN INPUT ERROR WAS DETECTED
       IF(ICK.NE.0) CALL EXIT
C
C      CALL SUBROUTINE OUT TO PRINT FAULT TABLE
       CALL OUT(ICK)
C
C      CALL SUBROUTINE MIN TO FIND MINIMAL EXPERIMENT
       CALL MIN(ICK)
C
C      IF ICK = 2, CIRCUIT IS REDUNDANT
       IF(ICK.EQ.2) CALL EXIT
C
C      CALL SUBROUTINE OUT TO PRINT MINIMAL EXPERIMENT
       CALL OUT(ICK)
       CALL EXIT
       END
```

```
C      SUBROUTINE INP IS THE CONTROLLING SUBROUTINE FOR
C      GENERATING THE FAULT TABLE
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        ICK    - RETURNS A 0 TO MAIN PROGRAM IF AN INPUT   *
C*                      ERROR OCURRED                        *
C*        IG     - NUMBER OF GATES                           *
C*        IN     - NUMBER OF INPUT VARIABLES                 *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE             *
C*        IV     - ARRAY OF INPUT VARIABLES                  *
C*        IO     - TYPE OF GATE                              *
C*        NG     - ARRAY OF GATES                            *
C*                 ITH GATE                                  *
C*                 NG(I,1) - GATE OUTPUT                     *
C*                 NG(I,2) - GATE TYPE                       *
C*                 NG(I,3) - NUMBER OF INPUTS TO GATE        *
C*                 NG(I,4) - NUMBER OF OUTPUTS FROM GATE     *
C*                 NG(I,5-12) - GATE INPUTS(MAX-8)           *
C*                 NG(I,13-20) - VALUE OF GATE INPUTS        *
C*                 NG(I,21) - COLUMN IN FAULT TABLE FOR      *
C*                            MULTIPLE OUTPUT GATE           *
C*                 NG(I,22-29) - COLUMN IN FAULT TABLE FOR   *
C*                               GATE INPUTS                 *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS               *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE          *
C*        S      - FAULT TABLE                               *
C*        LIM    - POSITION OF OUTPUT OF OUTPUT GATE IN      *
C*                      FAULT TABLE                          *
C*        LG     - ARRAY OF GATES TO BE WORKED WITH WHEN     *
C*                      GENERATING FAULT TABLE               *
C*        X      - LOGICAL .TRUE.                            *
C*        BLANK  - LOGICAL .FALSE.                           *
C*        KEEP   - HOLDS CORRECT VALUE OF INPUT OR OUTPUT    *
C*                      WIRE                                 *
C*************************************************************
      SUBROUTINE INP(ICK)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /BMAIN/NG(100,29),J,II(128)
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,NG,J,II,IF2,ICK
      LOGICAL*1 S,X/.TRUE./,BLANK/.FALSE./
      DATA IAND/'AND '/,IOR/'OR  '/,NOT/'NOT '/,
     1NAND/'NAND'/,NOR/'NOR '/
C
C     READ CARD 1 - NUMBER OF GATES AND NUMBER OF INPUT
C     VARIABLES
C
      READ(5,2000) IG,IN
 2000 FORMAT(2I3)
C
```

```
C       TERMINATE WITH ERROR MESSAGE IF NUMBER OF GATES IS < 1
C       OR > 100 OR NUMBER OF INPUT VARIABLES IS > 7 OR < 2
C
        IF(IG.LT.1.OR.IG.GT.100.OR.IN.GT.7.OR.IN.LT.2)
       1GO TO 1000
        I=2*IN
C
C       READ CARD 2 - INPUT VARIABLES AND COMPLEMENTS
C
        READ(5,2001) (IV(J),J=1,I)
 2001   FORMAT(14(1X,A2))
C
C       WRITE HEADINGS FOR CIRCUIT OUTPUT
C
        WRITE(6,2003)
 2003   FORMAT(1H1,53X,'THE COMBINATIONAL CIRCUIT' ,///,9X,
       1'GATE',8X,'GATE',5X,'NO  OF',5X,'NO  OF',6X,'1ST',7X,
       2'2ND',7X,'3RD',7X,'4TH',7X,'5TH',7X,'6TH',7X,'7TH',
       37X,'8TH',/,8X,'NUMBER',7X,'TYPE',5X,'INPUTS',5X,'OUT',
       4'PUTS',4X,'INPUT',5X,'INPUT',5X,'INPUT',5X,'INPUT',5X,
       5'INPUT',5X,'INPUT',5X,'INPUT',5X,'INPUT',//)
        DO 3 K=1,IG
        READ(5,2002) IO,(NG(K,J),J=3,12)
 2002   FORMAT(A4,2(2X,I2),8(2X,A2))
        NG(K,21)=0
        NZ=NG(K,3)+4
        WRITE(6,2004) K,IO,(NG(K,J),J=3,NZ)
 2004   FORMAT(1H ,8X,I3,9X,A4,8X,I1,9X,I2,1X,8(8X,A2))
C
C       IF NUMBER OF GATE INPUTS IS < 1 OR > 8 OR IF FANOUT OF
C       GATE IS < 1 OR > 10, TERMINATE WITH ERROR MESSAGE
C
        IF(NG(K,3).LT.1.OR.NG(K,3).GT.8.OR.NG(K,4).LT.1.OR.
       1NG(K,4).GT.10) GO TO 1000
        NG(K,2)=0
        IF(IO.EQ.IAND) NG(K,2)=1
        IF(IO.EQ.IOR) NG(K,2)=2
        IF(IO.EQ.NOT) NG(K,2)=3
        IF(IO.EQ.NAND) NG(K,2)=4
        IF(IO.EQ.NOR) NG(K,2)=5
C
C       IF GATE TYPE IS NOT AND, OR, NOT, NOR, NAND TERMINATE
C       WITH ERROR MESSAGE
C
        IF(NG(K,2).EQ.0) GO TO 1000
        IU=NG(K,3)
C
C       CONVERT GATE INPUTS TO APPROPRIATE CODE
C            EX:    A - 101
C                   B - 102
C                  A' - 103
```

```
C                B' - 104
C                GATE NUMBER(CHAR) - GATE NUMBER(NUMERIC)
C
      DO 3 I1=1,IU
      DO 6000 J=1,I
      IF(NG(K,I1+4).EQ.IV(J)) GO TO 7
 6000 CONTINUE
      CALL CONVT(NG(K,I1+4))
C
C     IF AN INCORRECT GATE INPUT IS DETECTED, TERMINATE
C     WITH ERROR MESSAGE
C
      IF(NG(K,I1+4).GE.100.OR.NG(K,I1+4).LE.0) GO TO 1000
      IF(NG(K,I1+4).GT.IG) GO TO 1000
      GO TO 3
    7 NG(K,I1+4)=100+J
    3 CONTINUE
C
C     IDENTIFY OUTPUT GATE
C
      WRITE(6,2005)
 2005 FORMAT(1H+,11X,'(OUTPUT)')
C
C     GENERATE INPUT COMBINATIONS
C
      K1=2**IN
      M=1
      J=IN
      DO 1 J1=1,K1,2
      INPUT(J1,J)=0
    1 INPUT(J1+1,J)=1
      IF(IN.EQ.1) GO TO 5
      M=M+1
      J=J-1
    4 L=(2**M)/2
      DO 2 J1=1,K1
      IF(J1.GT.L) GO TO 773
      INPUT(J1,J)=0
      K=0
      GO TO 2
  773 INPUT(J1,J)=1
      K=K+1
      IF(K.EQ.(2**(M-1))) L=L+2**M
    2 CONTINUE
      J=J-1
      M=M+1
      IF(J.GT.0) GO TO 4
C
C     FOR EACH INPUT COMBINATION, DETERMINE THE CORRECT
C     CIRCUIT OUTPUT AND POSSIBLE FAULTS THAT COULD
C     OCCUR IN CIRCUIT
```

```
C
    5 LIM=1
      DO 850 I=1,K1
      DO 114 J=1,IG
      NZ=NG(J,3)
C
C     FOR FIRST INPUT COMBINATION, GENERATE LOCATION OF
C     GATE INPUTS AND OUTPUTS IN FAULT TABLE
      IF(I.NE.1) GO TO 807
      DO 806 JJ=1,NZ
      IF(NG(J,JJ+4).GT.100) GO TO 100
      IF(NG(NG(J,JJ+4),4).EQ.1) GO TO 100
      IF(NG(NG(J,JJ+4),21).EQ.0) GO TO 99
      NG(J,21+JJ)=NG(NG(J,4+JJ),21)
      GO TO 806
   99 NG(NG(J,4+JJ),21)=LIM
      NG(J,21+JJ)=NG(NG(J,4+JJ),21)
      GO TO 101
  100 NG(J,21+JJ)=LIM
  101 LIM=LIM+1
  806 CONTINUE
C
C     GENERATE VALUE OF GATE INPUTS AND OUTPUTS
C
  807 DO 115 JJ=1,NZ
      IF(NG(J,4+JJ).LE.100) GO TO 805
      NW=NG(J,4+JJ)-100
      IF(NW.LE.IN) NG(J,12+JJ)=INPUT(I,NW)
      IF(NW.GT.IN) NG(J,12+JJ)=IABS(INPUT(I,NW-IN)-1)
      GO TO 115
  805 NG(J,12+JJ)=NG(NG(J,4+JJ),1)
  115 CONTINUE
C
C     CALL FUNCTION IGATE TO GENERATE GATE OUTPUT
C
  114 NG(J,1)=IGATE(NZ)
      IF(I.EQ.1) K=2*LIM
C
C     STORE CIRCUIT OUTPUT IN ARRAY IF1
C
      IF1(I)=NG(IG,1)
C
C     INITIALIZE EACH ROW OF FAULT TABLE WITH
C     BLANKS(.FALSE.)
C
      DO 117 IZ=1,K
  117 S(I,IZ)=BLANK
C
C     IF OUTPUT OF CIRCUIT IS S-A-0 OR S-A-1, PLACE
C     X(.TRUE.) IN APPROPRIATE COLUMN OF TABLE
C
```

```
      IF(NG(IG,1).EQ.1) S(I,LIM)=X
      IF(NG(IG,1).EQ.0) S(I,K)=X
      J=IG
      ICK=0
C
C     CALL SUBROUTINE TABLE TO COMPLETE GENERATION OF
C     FAULT TABLE
C
      CALL TABLE(LIM,I,ICK)
      IF(ICK.EQ.1) RETURN
C
C     CALL SUBROUTINE FANOUT TO CHECK ALL GATE OUTPUTS WITH
C     FANOUT > 1 FOR S-A-0 OR S-A-1
C
      CALL FANOUT(I,IG,LIM,ICK)
  850 CONTINUE
      RETURN
C
C     PRINT ERROR MESSAGE IF AN INPUT ERROR IS FOUND AND
C     SET INDICATOR(ICK)
C
 1000 WRITE(6,9821)
 9821 FORMAT(1HO,9X,'*****INPUT ERROR*****')
      ICK=1
      RETURN
      END
```

```
*     CALL CONVT(NG(K,I1+4)) - MAIN PROGRAM CALL SEQUENCE
*     NG(K,I1+4)=GATE INPUT WHICH IS NOT AN INPUT VARIABLE
*          AND IS TO BE CONVERTED FROM CHARACTER TO NUMERIC
*
CONVT     START   0
          SAVE    (14,12),T,*
          BALR    12,0          R12 IS BASE REGISTER
          USING   *,12
          L       7,0(0,1)      C(R7) = ADDRESS OF NG(K,I1+4)
          LH      4,0(0,7)      C(R4) = VALUE OF NG(K,I1+4)
          SR      5,5           INITIALIZE R5 = 0
          SRDL    4,4           SHIFT VALUE OF ONES DIGIT
          SRL     5,28               INTO R5
          ST      5,HOLD        STORE ONES DIGIT IN HOLD
          SRDL    4,4           SHIFT ZONE BITS OF ONES DIGIT
          SRL     5,28               INTO R5
          C       5,=F'15'      CHECK FOR NUMERIC ZONE
          BC      8,GOON        IF NONNUMERIC ZONE SET
          SR      5,5                R5 = 0 AND
          BC      15,RTN             BRANCH TO RTN
GOON      SRDL    4,4           SHIFT TENS DIGIT INTO R5
          N       4,=X'0000000F'     ZONE BITS IN R4
          C       4,=F'15'      CHECK FOR NUMERIC ZONE
          BE      GONE          BRANCH IF NUMERIC ZONE
          C       4,=F'4'       CHECK FOR SPACE ZONE
          BE      GONE          BRANCH IF SPACE ZONE
          SR      5,5                R5 = 0
          BC      15,RTN        BRANCH TO RTN
GONE      SRL     5,28          TENS DIGIT IN R5
          M       4,=F'10'      MULTIPLY TENS DIGIT(R5) BY 10
          A       5,HOLD        ADD TENS DIGIT TO ONES DIGIT
RTN       STH     5,0(0,7)      STORE VALUE BACK IN NG(K,I1+4)
          RETURN  (14,12),T     RETURN
          DS      0F
HOLD      DS      F
          END
```

```
C      FUNCTION IGATE EXAMINES GATE INPUTS AND RETURNS
C      VALUE OF GATE OUTPUT
       FUNCTION IGATE(NZ)
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        NG - ARRAY OF GATES                                *
C*        J  - GATE NUMBER                                   *
C*        NZ - NUMBER OF INPUTS FOR GATE                     *
C*        IZ - GATE TYPE                                     *
C*             1-AND, 2-OR, 3-NOT, 4-NAND, 5-NOR             *
C*************************************************************
C
       COMMON /BMAIN/NG(100,29),J,II(128)
       INTEGER*2 NG,IZ,I,J,II
C
C      BRANCH TO APPROPRIATE GATE TYPE
C
       IZ=NG(J,2)
       GO TO (7,4,3,7,4),IZ
C
C      AND OR NAND GATE - EXAMINE INPUTS, IF AN INPUT = 0
C         FOR AND, OUTPUT = 0     FOR NAND, OUTPUT IS 1:
C         IF ALL INPUTS ARE 1 FOR AND, OUTPUT = 1   FOR NAND
C         OUTPUT = 0
     7 DO 1 I=1,NZ
       IF(NG(J,12+I).EQ.0) GO TO 2
     1 CONTINUE
     5 IGATE=1
       IF(NG(J,2).EQ.4.OR.NG(J,2).EQ.5) IGATE=0
       RETURN
     2 IGATE=0
       IF(NG(J,2).EQ.4.OR.NG(J,2).EQ.5) IGATE=1
       RETURN
C
C         NOT GATE - OUTPUT = OPPOSITE OF INPUT
     3 IGATE=IABS(NG(J,12+1)-1)
       RETURN
C
C      OR, NOR GATE - EXAMINE INPUTS, IF AN INPUT = 1 FOR OR
C         OUTPUT = 1   FOR NOR, OUTPUT = 0:  IF ALL INPUTS
C         = 0 FOR OR, OUTPUT = 0   FOR NOR, OUTPUT = 1
C
     4 DO 6 I=1,NZ
       IF(NG(J,12+I).EQ.1) GO TO 5
     6 CONTINUE
       GO TO 2
       END
```

```
C      SUBROUTINE NGATE EXAMINES INPUTS OF JTH GATE BEGINNING
C      AT IOTH INPUT UNTIL AN INPUT CAN BE DETECTED S-A-0 OR
C      S-A-1
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        NG     - ARRAY OF GATES AS DESCRIBED IN SUB INP    *
C*        J      - GATE NUMBER                               *
C*        IO     - GATE INPUT NUMBER                         *
C*        S      - FAULT TABLE                               *
C*        II     - ARRAY TO KEEP TRACK OF ALL GATES ON A     *
C*                      PATH BEING CHECKED                   *
C*        X      - LOGICAL .TRUE.                            *
C*        BLANK  - LOGICAL .FALSE.                           *
C*        I      - ROW OF TABLE                              *
C*        LIM    - POSITION OF CIRCUIT OUTPUT IN FAULT TABLE *
C*************************************************************
      SUBROUTINE NGATE(IO,I2,I8,LIM,I)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /BMAIN/NG(100,29),J,II(128)
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,NG,J,II,IF2
      LOGICAL*1 S,X/.TRUE./,BLANK/.FALSE./
      IZ=NG(J,2)
      GO TO (121,122,123,121,122),IZ
  121 IF(NG(J,2).EQ.1.AND.NG(J,1).EQ.0) GO TO 225
      IF(NG(J,2).EQ.4.AND.NG(J,1).EQ.1) GO TO 225
C
C     GATE J IS AND GATE WITH OUTPUT OF 1 OR NAND GATE WITH
C     OUTPUT OF 0
      S(I,NG(J,21+IO))=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=0
      RETURN
C
C     GATE J IS AND GATE WITH OUTPUT OF 0 OR NAND GATE WITH
C     OUTPUT OF 1
C
  225 IU=NG(J,3)
      I4=0
      DO 226 I3=1,IU
      IF(NG(J,12+I3).EQ.0) I4=I4+1
      IF(I4.GT.1) GO TO 224
      IF(NG(J,12+I3).EQ.0) IO=I3
  226 CONTINUE
      S(I,NG(J,21+IO)+LIM)=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=1
      RETURN
  122 IF(NG(J,2).EQ.2.AND.NG(J,1).EQ.1) GO TO 222
```

```
      IF(NG(J,2).EQ.5.AND.NG(J,1).EQ.0) GO TO 222
C
C     GATE J IS OR GATE WITH OUTPUT OF 0 OR NOR GATE WITH
C     OUTPUT OF 1
C
      S(I,NG(J,21+IO)+LIM)=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=0
      RETURN
C
C     GATE J IS OR GATE WITH OUTPUT OF 1 OR NOR GATE WITH
C     OUTPUT OF 0
C
  222 IU=NG(J,3)
      I4=0
      DO 223 I3=1,IU
      I4=I4+NG(J,12+I3)
      IF(I4.GT.1) GO TO 224
      IF(NG(J,12+I3).EQ.1) IO=I3
  223 CONTINUE
      S(I,NG(J,21+IO))=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=1
      RETURN
C
C     AND, NAND GATE WITH 2 OR MORE INPUTS 0 OR AN OR, NOR
C     GATE WITH 2 OR MORE INPUTS 1 AND NO INPUT FAULT CAN BE
C     DETECTED
C
  224 I8=1
      RETURN
  123 IF(NG(J,1).EQ.1) GO TO 227
C
C     NOT GATE WITH OUTPUT OF 0
C
      S(I,NG(J,21+IO))=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=1
      RETURN
C
C     NOT GATE WITH OUTPUT OF 1
C
  227 S(I,NG(J,21+IO)+LIM)=X
      II(I2)=J
      II(I2+1)=IO
      II(I2+2)=1
      RETURN
      END
```

```
C      SUBROUTINE TABLE CONTROLS INDIVIDUAL GATES AS THEY ARE
C      CHECKED FOR FAULTS
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        I8     - A 1 INDICATES A FAULT CANNOT BE DETECTED  *
C*                 FOR GATE INPUTS                           *
C*        IO     - INDICATES WHICH INPUT OF A GATE IS BEING  *
C*                 CHECKED                                   *
C*        II     - ARRAY TO KEEP TRACK OF ALL GATES ON A     *
C*                 PATH BEING CHECKED                        *
C*                 II(I2) - GATE NUMBER                      *
C*                 II(I2+1) - GATE INPUT NUMBER              *
C*                 II(I2+2) - A 1 INDICATES ONLY ONE         *
C*                   INPUT OF GATE CAN BE S-A-0 OR S-A-1     *
C*        LIM    - POSITION OF CIRCUIT OUTPUT IN FAULT TABLE *
C*        J      - GATE NUMBER                               *
C*        I      - ROW OF TABLE                              *
C*        NG     - ARRAY OF GATES AS DESCRIBED IN SUBROUTINE *
C*                 INP                                       *
C*************************************************************
      SUBROUTINE TABLE(LIM,I,ICK)
      COMMON /BMAIN/NG(100,29),J,II(128)
      INTEGER*2 NG,J,II,ICK
      I8=0
      IO=1
      I2=-2
  969 I2=I2+3
C
C     IF LEVEL OF GATING IN CIRCUIT EXCEEDS 42, TERMINATE
C     WITH ERROR MESSAGE
C
      IF(I2.GT.126) GO TO 1000
C
C     CALL SUBROUTINE NGATE TO EXAMINE INPUTS OF JTH GATE
C     BEGINNING AT IOTH INPUT
C
  970 CALL NGATE(IO,I2,I8,LIM,I)
C
C     IF I8=1, A FAULT CAN'T BE DETECTED FOR INPUTS OF
C     GATE J
C
      IF(I8.NE.1) GO TO 991
      I8=0
C
C     PROCEED BACK ALONG PATH TOWARD OUTPUT GATE UNTIL
C     OUTPUT GATE IS REACHED OR ANOTHER PATH CAN BE FOLLOWED
C
  993 I2=I2-3
      IF(I2.LT.1) GO TO 113
      IO=II(I2+1)+1
```

```
      IF(II(I2+2).EQ.1) GO TO 993
      IF(IO.LE.NG(II(I2),3)) GO TO 992
      GO TO 993
  992 J=II(I2)
      GO TO 970
C
C     BACKTRACK FROM OUTPUT GATE THROUGH CIRCUITS UNTIL
C     INPUTS TO CIRCUIT ARE REACHED
C
  991 IF(NG(II(I2),4+II(I2+1)).GT.100) GO TO 990
      J=NG(II(I2),4+II(I2+1))
      IO=1
      GO TO 969
  990 IO=IO+1
      IF(IO.GT.NG(II(I2),3).OR.II(I2+2).EQ.1) GO TO 993
      GO TO 970
  113 RETURN
C
C     PRINT ERROR MESSAGE IF LEVEL OF GATING EXCEEDS 42
C     AND SET INDICATOR
C
 1000 WRITE(6,1001)
 1001 FORMAT(1HO,'***ERROR-LEVEL OF GATING EXCEEDS 42***')
      ICK=1
      RETURN
      END
```

```
C      SUBROUTINE FANOUT EXAMINES ALL GATE OUTPUTS WITH
C      FANOUT > 1 FOR POSSIBLE FAULTS NOT DETECTED BY
C       SUBROUTINE TABLE
C
C*****************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                         *
C*        IG     - NUMBER OF GATES                               *
C*        I      - ROW OF TABLE                                  *
C*        LIM    - POSITION OF OUTPUT OF CIRCUIT IN FAULT        *
C*                      TABLE                                    *
C*        NG     - ARRAY OF GATES AS DESCRIBED IN SUBROUTINE     *
C*                      INP                                      *
C*        KEEP   - VARIABLE TO STORE CORRECT GATE OUTPUT         *
C*        X      - LOGICAL .TRUE.                                *
C*        BLANK  - LOGICAL .FALSE.                               *
C*        IF1    - ARRAY OF CORRECT CIRCUIT OUTPUTS              *
C*****************************************************************
      SUBROUTINE FANOUT(I,IG,LIM,ICK)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /BMAIN/NG(100,29),J,II(128)
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,NG,J,II,IF2,ICK
      LOGICAL*1 S,X/.TRUE./,BLANK/.FALSE./
      I3=IG-1
      DO 114 J1=1,I3
C
C     IF GATE J1 HAS FANOUT > 1, CHECK FOR IT BEING S-A-0 OR
C     S-A-1 AND ALL NODES LEADING TO THIS GATE FOR S-A-0 OR
C     S-A-1
C
      IF(NG(J1,4).EQ.1) GO TO 114
      KEEP=NG(J1,1)
      NG(J1,1)=IABS(NG(J1,1)-1)
      I4=J1+1
      DO 115 J=I4,IG
      IU=NG(J,3)
      DO 116 J2=1,IU
      IF(NG(J,4+J2).LE.100) NG(J,12+J2)=NG(NG(J,4+J2),1)
  116 CONTINUE
C
C     CALL FUNCTION IGATE TO GENERATE OUTPUT OF GATE J
C
  115 NG(J,1)=IGATE(IU)
C
C     MAKE APPROPRIATE COLUMNS BLANK(.FALSE.)
C
      S(I,NG(J1,21))=BLANK
      S(I,NG(J1,21)+LIM)=BLANK
C
C     IF OUTPUT OF GATE J1 IS S-A-0 OR S-A-1 PLACE AN
C     X(.TRUE.) IN APPROPRIATE COLUMN OF TABLE
```

```
C
      IF(KEEP.EQ.0.AND.NG(IG,1).NE.IF1(I)) S(I,NG(J1,21)+LIM
     1)=X
      IF(KEEP.EQ.1.AND.NG(IG,1).NE.IF1(I)) S(I,NG(J1,21))=X
      NG(J1,1)=KEEP
      KEEP=NG(IG,1)
      I4=J1+1
C
C     RESTORE GATES FROM J1+1 TO IG
C
      DO 117 J=I4,IG
      IU=NG(J,3)
      DO 118 J2=1,IU
      IF(NG(J,4+J2).LE.100) NG(J,12+J2)=NG(NG(J,4+J2),1)
  118 CONTINUE
  117 NG(J,1)=IGATE(IU)
      IF(KEEP.EQ.IF1(I)) GO TO 114
      J=J1
C
C     IF OUTPUT OF GATE J1 IS S-A-0 OR S-A-1, CALL TABLE
C     AND CHECK ALL NODES LEADING TO GATE J1
C
      CALL TABLE(LIM,I,ICK)
  114 CONTINUE
      RETURN
      END
```

```
C      SUBROUTINE OUT OUTPUTS THE FAULT TABLE AND THE MINIMAL
C      EXPERIMENT
C
C*****************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                         *
C*        IN     - NUMBER OF INPUT VARIABLES                     *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE              *
C*        L      - END NODE FOR SECTION OF TABLE                 *
C*        ICOL   - ARRAY USED TO OUTPUT NODE LABELS AND A        *
C*                     ROW OF TABLE                              *
C*        IS     - ARRAY OF INPUT VARIABLES                      *
C*        ICK    - VARIABLE TO INDICATE WHETHER FAULT TABLE      *
C*                     OR MINIMAL EXPERIMENT IS TO BE OUTPUT     *
C*                     AND WHETHER SECTION IS S-A-0 OR S-A-1     *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS                   *
C*        S      - FAULT TABLE                                   *
C*        X      - LITERAL 'X'                                   *
C*        BLANK  - LITERAL ' '                                   *
C*        IF1    - ARRAY OF CORRECT OUTPUTS                      *
C*        M      - START NODE FOR A SECTION OF TABLE             *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE                 *
C*        J1     - NUMBER OF FAULT TESTS IN MINIMAL              *
C*                     EXPERIMENT                                *
C*        NUT    - ARRAY OF INPUT COMBINATIONS IN MINIMAL        *
C*                     EXPERIMENT                                *
C*        IF2    - ARRAY OF INCORRECT OUTPUTS IN MINIMAL         *
C*                     EXPERIMENT                                *
C*****************************************************************
      SUBROUTINE OUT(ICK)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /DMAIN/ICOL(1600),J8
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,ICOL,X/'X '/,
     1BLANK/'  '/,ICK,IF2
      LOGICAL*1 S,IS(12)
      EQUIVALENCE (IS(1),IV(1))
C
C     IF ICK = 1, PRINT MINIMAL EXPERIMENT, OTHERWISE
C     PRINT FAULT TABLE
C
      IF(ICK.EQ.1) GO TO 9998
      J8=2*IN
      L=K/2
      DO 301 J=1,L
  301 ICOL(J)=J
      IF(K.GT.84) L=42
      IF(K.GT.44) GO TO 1800
C
C     OUTPUT FAULT TABLE WHEN LESS THAN 44 COLUMNS
C
      WRITE(6,300)
```

```
  300 FORMAT(1H1,58X,'THE FAULT TABLE',//,1X,'************',
     1'**************************************************',
     2'**************************************************',
     3'******************')
      WRITE(6,3100)
 3100 FORMAT(1H ,'INPUTS *CORRECT*',18X,'POSSIBLE S-A-0 FA',
     1'ULTS',18X,'*',19X,'POSSIBLE S-A-1 FAULTS',/,8X,
     2'*OUTPUT*')
      WRITE(6,302) (ICOL(J),J=1,L)
  302 FORMAT(1H+,15X,9I2,13I3)
      WRITE(6,303) (ICOL(J),J=1,L)
  303 FORMAT(1H+,73X,'*',9I2,13I3)
      DO 304 J=1,L
  304 ICOL(J)=0
      WRITE(6,305) (ICOL(J),J=1,L)
  305 FORMAT(1H ,7X,'*  F   *',9I2,13I3)
      DO 306 J=1,L
  306 ICOL(J)=1
      WRITE(6,307) (ICOL(J),J=1,L)
  307 FORMAT(1H+,73X,'*',9I2,13I3)
      WRITE(6,321) (IS(J),J=2,J8,2)
  321 FORMAT(1H+,7A1)
      WRITE(6,331)
  331 FORMAT(1H ,'****************************************',
     1'****************************************',
     2'****************************************',/)
      DO 313 I=1,K1
      WRITE(6,392) (INPUT(I,J),J=1,IN)
  392 FORMAT(1H ,7I1)
C
C     CONVERT ROW OF FAULT TABLE TO X'S AND BLANK'S IN
C     ICOL FOR OUTPUT
C
      DO 400 J=1,K
      IF(S(I,J)) GO TO 401
      ICOL(J)=BLANK
      GO TO 400
  401 ICOL(J)=X
  400 CONTINUE
      WRITE(6,394) IF1(I),(ICOL(J),J=1,L)
  394 FORMAT(1H+,10X,I1,5X,A1,8(1X,A1),13(2X,A1))
      M=L+1
  313 WRITE(6,395) (ICOL(J),J=M,K)
  395 FORMAT(1H+,75X,A1,8(1X,A1),13(2X,A1))
      ICK=1
      RETURN
C
C     OUTPUT FAULT TABLE WHEN GREATER THAN 44 COLUMNS
C
 1800 M=1
      IL=0
```

```
      ICK=0
      WRITE(6,300)
 1809 WRITE(6,1801) ICK
 1801 FORMAT(1H ,'INPUTS *CORRECT*',40X,'POSSIBLE S-A-',I1,
     1' FAULTS',/,8X,'*OUTPUT*')
      WRITE(6,1802) (ICOL(J),J=M,L)
 1802 FORMAT(1H+,15X,9I2,33I3)
      DO 1812 J=M,L
 1812 ICOL(J)=ICK
      WRITE(6,1813) (ICOL(J),J=M,L)
 1813 FORMAT(1H ,7X,'*  F   *',9I2,33I3)
      WRITE(6,1814) (IS(J),J=2,J8,2)
 1814 FORMAT(1H+,7A1)
      WRITE(6,331)
      DO 1806 I=1,K1
      WRITE(6,392) (INPUT(I,J),J=1,IN)
C
C     CONVERT ROW OF FAULT TABLE TO X'S AND BLANK'S IN
C     ICOL FOR OUTPUT
C
      DO 1803 J=M,L
      IF(S(I,J+IL)) GO TO 1804
      ICOL(J)=BLANK
      GO TO 1803
 1804 ICOL(J)=X
 1803 CONTINUE
 1806 WRITE(6,1808) IF1(I),(ICOL(J),J=M,L)
 1808 FORMAT(1H+,10X,I1,5X,A1,8(1X,A1),33(2X,A1))
      IF(K.GT.84) GO TO 1900
      IF(ICK.EQ.1) RETURN
      WRITE(6,1805)
 1805 FORMAT(1H0,///,1X,58X,'THE FAULT TABLE(CONTINUED)',//,
     11X,'*********************************************',
     2'*********************************************',
     3'**********************************')
      ICK=1
      IL=K/2
      DO 1807 J=1,L
 1807 ICOL(J)=J
      GO TO 1809
C
C     CALL CONT TO OUTPUT A SECTION OF TABLE WHEN NUMBER OF
C     COLUMNS IS GREATER THAN 84
C
 1900 M=L+1
      IF(ICK.EQ.1) IL=K/2
      IF(K.LE.162) L=K/2
      IF(K.GT.162) L=81
 1902 CALL CONT(ICK,M,L,IL,IS)
      IF(K.GT.162) GO TO 1901
      IF(ICK.EQ.1) RETURN
```

```
 1906 M=1
      L=42
      ICK=1
      WRITE(6,1805)
      DO 1828 J=1,L
 1828 ICOL(J)=J
      IL=K/2
      GO TO 1809
 1901 IF(L.EQ.(K/2)) GO TO 1905
      M=L+1
      IF(L.EQ.115) L=L+29
      IF(L.EQ.81) L=115
      IF(L.GT.(K/2)) L=K/2
      GO TO 1902
 1905 IF(ICK.EQ.1) RETURN
      GO TO 1906
C
C     OUTPUT MINIMAL EXPERIMENT
C
 9998 WRITE(6,113)
  113 FORMAT(1H1,32X,'A MINIMAL FAULT-DETECTION EXPERIMENT',
     1' CONSISTS OF THE FOLLOWING TESTS',//)
      J1=K1
      I1=J1-1
      IF(J1.EQ.1) GO TO 143
      DO 141 J7=1,I1
      WRITE(6,116) J7,(NUT(J7,I),I=1,IN)
  116 FORMAT(1H0,5X,I2,') APPLY ',7I1)
  141 WRITE(6,115) IF2(J7)
  115 FORMAT(1H+,22X,': IF THE OUTPUT IS F = ',I1,', CIRCU',
     1'IT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTH',
     2'ERWISE, APPLY NEXT TEST')
  143 WRITE(6,116) J1,(NUT(J1,I),I=1,IN)
      WRITE(6,160) IF2(J1)
  160 FORMAT(1H+,22X,': IF F = ',I1,', CIRCUIT IS FAULTY; ',
     1'OTHERWISE, CIRCUIT OPERATES CORRECTLY')
      RETURN
      END
```

```
C      SUBROUTINE CONT OUTPUTS FAULT TABLE FROM 2ND SECTION
C      ON, WHEN FAULT TABLE CONTAINS MORE THAN 44 COLUMNS
C
C*****************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                         *
C*        ICK    - VARIABLE TO DENOTE S-A-1 OR S-A-0             *
C*        ICOL   - ARRAY USED TO OUTPUT NODE LABELS AND A        *
C*                       ROW OF TABLE                            *
C*        M      - STARTING NODE FOR SECTION OF TABLE            *
C*        L      - END NODE FOR SECTION OF TABLE                 *
C*        IL     - EITHER BEGINNING(0) OF TABLE OR               *
C*                       MIDDLE(K/2) OF FAULT TABLE              *
C*        IS     - ARRAY OF INPUT VARIABLES                      *
C*        S      - FAULT TABLE                                   *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS                   *
C*        IF1    - ARRAY OF CORRECT OUTPUTS                      *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE              *
C*        K1     - NUMBER OF ROWS IN TABLE                       *
C*        BLANK  - LITERAL ' '                                   *
C*        X      - LITERAL 'X'                                   *
C*****************************************************************
C
      SUBROUTINE CONT(ICK,M,L,IL,IS)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /DMAIN/ICOL(1600),J8
      LOGICAL*1 S,IS(14)
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,ICK,ICOL,
     1X/'X '/,BLANK/'  '/
      WRITE(6,330)
  330 FORMAT(1HO,///,1X,'**********************************',
     1'**************************************************',
     2'**********************************************')
      WRITE(6,1910) ICK
 1910 FORMAT(1H ,'INPUTS *CORRECT*',40X,'POSSIBLE S-A-',I1,
     1' FAULTS(CONTINUED)',/,8X,'*OUTPUT*')
      DO 1911 J=M,L
 1911 ICOL(J)=J
      IF(L.GT.42.AND.L.LE.81) WRITE(6,2) (ICOL(J),J=M,L)
    2 FORMAT(1H+,15X,I2,38I3)
      IF(L.GT.81.AND.L.LE.115) WRITE(6,12) (ICOL(J),J=M,L)
   12 FORMAT(1H+,15X,I2,17I3,16I4)
      IF(L.GT.115) WRITE(6,13) (ICOL(J),J=M,L)
   13 FORMAT(1H+,15X,I3,28I4)
      DO 3 J=M,L
    3 ICOL(J)=ICK
      IF(L.GT.42.AND.L.LE.81) WRITE(6,22) (ICOL(J),J=M,L)
   22 FORMAT(1H ,7X,'*  F    *',I2,38I3)
      IF(L.GT.81.AND.L.LE.115) WRITE(6,23) (ICOL(J),J=M,L)
   23 FORMAT(1H ,7X,'*  F    *',I2,18I3,15I4)
      IF(L.GT.115) WRITE(6,24) (ICOL(J),J=M,L)
```

```
   24 FORMAT(1H ,7X,'*  F   *',I2,28I4)
      WRITE(6,1814) (IS(J),J=2,J8,2)
 1814 FORMAT(1H+,7A1)
      WRITE(6,331)
  331 FORMAT(1H ,'*****************************************',
     1'*****************************************',
     2'*****************************************',/)
      DO 5 I=1,K1
      WRITE(6,392) (INPUT(I,J),J=1,IN)
  392 FORMAT(1H ,7I1)
      DO 6 J=M,L
      IF(S(I,J+IL)) GO TO 7
      ICOL(J)=BLANK
      GO TO 6
    7 ICOL(J)=X
    6 CONTINUE
      IF(L.GT.42.AND.L.LE.81) WRITE(6,4) IF1(I),
     1(ICOL(J),J=M,L)
    4 FORMAT(1H+,10X,I1,5X,A1,38(2X,A1))
      IF(L.GT.81.AND.L.LE.115) WRITE(6,25) IF1(I),
     1(ICOL(J),J=M,L)
   25 FORMAT(1H+,10X,I1,5X,A1,17(2X,A1),2X,A1,15(3X,A1))
      IF(L.GT.115) WRITE(6,26) IF1(I),(ICOL(J),J=M,L)
   26 FORMAT(1H+,10X,I1,5X,A1,28(3X,A1))
    5 CONTINUE
      RETURN
      END
```

```
C      SUBROUTINE MIN IS THE CONTROLLING SUBROUTINE FOR
C      REDUCING THE TABLE
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        ICK    - RETURNS A 2 TO MAIN PROGRAM IF CIRCUIT IS *
C*                        REDUNDANT                          *
C*        FIRST  - LOGICAL VARIABLE NECESSARY FOR            *
C*                  DETERMINING COMBINATIONS IF TABLE IS     *
C*                  CYCLIC                                   *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE            *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE         *
C*        J1     - NUMBER OF TESTS IN MINIMAL EXPERIMENT    *
C*        ICOL   - ARRAY CONTAINING NUMBER OF FAULTS PER COL *
C*        S      - FAULT TABLE                              *
C*        IROW   - ARRAY CONTAINING NUMBER OF FAULTS PER ROW *
C*        IN     - NUMBER OF INPUT VARIABLES                *
C*        IF1    - ARRAY OF CORRECT OUTPUTS                 *
C*        IF2    - ARRAY OF INCORRECT OUTPUTS IN MINIMAL    *
C*                   EXPERIMENT                            *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS              *
C*        NUT    - ARRAY OF INPUT COMBINATIONS IN MINIMAL   *
C*                    EXPERIMENT                           *
C*        II     - ARRAY OF COMBINATIONS OF K1 ITEMS TAKEN  *
C*                     IR AT A TIME                        *
C*************************************************************
      SUBROUTINE MIN(ICK)
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      COMMON /CMAIN/K2,I1,J,IROW(128),ICOL(1600),II(128)
      LOGICAL*1 S
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,ICK,IF2,K2,I1,J,II,
     1ICOL,IROW
      LOGICAL FIRST/.TRUE./
      N=K1
      J1=0
      K2=K
C
C     CALL RECNT TO COUNT NUMBER OF X'S IN EACH ROW AND
C     COLUMN OF FAULT TABLE
C
      CALL RECNT
C
C     IF ANY COLUMN DOESN'T HAVE ANY X'S THEN CIRCUIT IS
C     REDUNDANT
C
      DO 40 J=1,K
      IF(ICOL(J).EQ.0) GO TO 58
   40 CONTINUE
      L1=0
C
```

```
C       IF ANY COLUMN HAS A SINGLE X, THE INPUT COMBINATION
C       FOR THE ROW CONTAINING THE X IS AN ESSENTIAL PRIME
C       IMPLICANT
C
   31   DO 33 J=1,K
        IF(ICOL(J).NE.1) GO TO 33
        DO 32 I1=1,K1
        IF(S(I1,J)) GO TO 34
   32   CONTINUE
   33   CONTINUE
        GO TO 36
   34   J1=J1+1
C
C       A PRIME IMPLICANT HAS BEEN FOUND AND IS ADDED TO THE
C       MINIMAL FAULT-DETECTION EXPERIMENT
C
        DO 13 I=1,IN
        IF2(J1)=IABS(IF1(I1)-1)
   13   NUT(J1,I)=INPUT(I1,I)
        IF(K1.EQ.1) GO TO 43
        K2=K
        L1=0
C
C       CALL REDUCE TO REMOVE PRIME IMPLICANT FROM THE TABLE
C
        CALL REDUCE
        IF(K.EQ.0) GO TO 43
        GO TO 31
   36   IF(K1.EQ.N) GO TO 42
        IF(L1.EQ.1) GO TO 102
   42   JC=K1
C
C       CALL ROWS TO REMOVE DOMINATED ROWS
C
        CALL ROWS
        IF(JC.EQ.K1) GO TO 102
C
C       CALL RECNT TO COUNT NUMBER OF X'S IN EACH ROW AND
C       COLUMN OF FAULT TABLE
C
        CALL RECNT
        IF(L1.NE.0) GO TO 102
        L1=1
        GO TO 31
  102   JC=K
C
C       CALL COLUMN TO REMOVE DOMINATING COLUMNS
C
        CALL COLUMN
        IF(JC.EQ.K) GO TO 44
C
```

```
C      CALL RECNT TO COUNT NUMBER OF X'S IN EACH ROW AND
C      COLUMN OF FAULT TABLE
C
       CALL RECNT
       L1=0
       GO TO 42
   44  N1=K1-1
C
C      IF TABLE IS CYCLIC, BEGIN WITH IR=2 AND CHECK FOR
C      COMBINATIONS OF K1 ITEMS TAKEN IR AT A TIME UNTIL A
C      COVER FOR THE CYCLIC TABLE IS FOUND
C
       DO 190 IR=2,N1
  223  CALL COMB(FIRST,K1,IR,II)
       IF(FIRST) GO TO 190
       IS=0
       DO 222 J=1,IR
  222  IS=IS+IROW(II(J))
       IF(IS.LT.K) GO TO 223
       DO 242 J=1,K
       DO 266 J2=1,IR
       IF(S(II(J2),J)) GO TO 242
  266  CONTINUE
       GO TO 223
  242  CONTINUE
       GO TO 272
  190  CONTINUE
       IR=K1
  272  DO 282 I1=1,IR
       J1=J1+1
       IF(IR.EQ.K1) II(I1)=I1
       DO 282 J2=1,IN
       IF2(J1)=IABS(IF1(II(I1))-1)
  282  NUT(J1,J2)=INPUT(II(I1),J2)
   43  ICK=1
       K1=J1
       RETURN
C
C      PRINT MESSAGE DENOTING REDUNDANT CIRCUIT
C
   58  IF(J.GT.(K/2)) J=J-(K/2)
       WRITE(6,22) J
   22  FORMAT(1H0,///,1X,'***CIRCUIT IS REDUNDANT(COLUMN ',
      1I4,')***')
       ICK=2
       RETURN
       END
```

```
C      SUBROUTINE RECNT COUNTS THE NUMBER OF FAULTS(X'S) IN
C      EACH ROW AND COLUMN OF THE FAULT TABLE
C
C*****************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                         *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE                 *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE              *
C*        S      - FAULT TABLE                                   *
C*        IROW   - ARRAY CONTAINING NUMBER OF FAULTS PER ROW     *
C*        ICOL   - ARRAY CONTAINING NUMBER OF FAULTS PER COL     *
C*****************************************************************
C
      SUBROUTINE RECNT
      COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
     1NUT(128,7),IV(14),S(128,1600)
      LOGICAL*1 S
      COMMON /CMAIN/K2,I1,J,IROW(128),ICOL(1600),II(128)
      INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,K2,I1,J,ICOL,
     1IROW,II
      DO 19 I=1,K1
      J2=0
      DO 29 JI=1,K
      IF(S(I,JI)) J2=J2+1
   29 CONTINUE
   19 IROW(I)=J2
      DO 89 JI=1,K
      J2=0
      DO 88 I=1,K1
      IF(S(I,JI)) J2=J2+1
   88 CONTINUE
   89 ICOL(JI)=J2
      RETURN
      END
```

```
C      SUBROUTINE COLUMN REMOVES DOMINATING COLUMNS FROM THE
C      FAULT TABLE
C
C*****************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                         *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE              *
C*        I3     - STARTING POINT TO CHECK FOR                   *
C*                        DOMINATING COLUMNS                     *
C*        ICOL   - ARRAY CONTAINING NUMBER OF FAULTS PER COL     *
C*        S      - FAULT TABLE                                   *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE                 *
C*****************************************************************
       SUBROUTINE COLUMN
       COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
      1NUT(128,7),IV(14),S(128,1600)
       LOGICAL*1 S
       COMMON /CMAIN/K2,I1,J,IROW(128),ICOL(1600),II(128)
       INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,K2,I1,J,ICOL,
      1IROW,II
       I3=1
     7 J1=K
       IF(I3.GE.K) RETURN
       DO 1 I=I3,K
       IF(I.EQ.K) RETURN
       I8=I+1
       DO 1 J=I8,K
C
C      COMPARE THE NUMBER OF X'S IN ICOL(I) AND ICOL(J)
C      IF ICOL(I) = ICOL(J), REMOVE ICOL(J)
C
       IF(ICOL(I).NE.ICOL(J)) GO TO 4
       DO 2 I1=1,K1
       IF((.NOT.S(I1,J).AND.S(I1,I)).OR.(S(I1,J).AND..NOT.
      1S(I1,I))) GO TO 1
     2 CONTINUE
       GO TO 8
     4 IF(ICOL(I).LT.ICOL(J)) GO TO 5
C
C      IF ICOL(I) DOMINATES ICOL(J), REMOVE ICOL(I)
C
       DO 3 I1=1,K1
       IF(.NOT.S(I1,J)) GO TO 3
       IF(.NOT.S(I1,I)) GO TO 1
     3 CONTINUE
C
C      INITIALIZE VARIABLES TO REMOVE ICOL(I)
C
     8 K=I-1
       I3=I
       GO TO 9
C
```

```
C       IF ICOL(J) DOMINATES ICOL(I), REMOVE ICOL(J)
C
    5 DO 6 I1=1,K1
      IF(.NOT.S(I1,I)) GO TO 6
      IF(.NOT.S(I1,J)) GO TO 1
    6 CONTINUE
C
C       INITIALIZE VARIABLES TO REMOVE ICOL(J)
C
      K=J-1
      I3=I
      GO TO 9
    1 CONTINUE
      RETURN
    9 K=K+1
      IF(K.NE.J1) GO TO 10
      I3=I3+1
      K=K-1
      GO TO 7
C
C       REMOVE APPROPRIATE COLUMN FROM ARRAY, ICOL, AND FROM
C       FAULT TABLE
C
   10 ICOL(K)=ICOL(K+1)
      DO 11 I1=1,K1
   11 S(I1,K)=S(I1,K+1)
      IF(K.NE.J1-1) GO TO 9
      GO TO 7
      END
```

```
C      SUBROUTINE REDUCE IS USED TO REMOVE A PRIME IMPLICANT
C      FROM THE FAULT TABLE
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        K2     - NUMBER OF COLUMNS IN FAULT TABLE          *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE          *
C*        S      - FAULT TABLE                               *
C*        ICOL   - ARRAY CONTAINING NUMBER OF FAULTS PER COL *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE             *
C*        IROW   - ARRAY CONTAINING NUMBER OF FAULTS PER ROW *
C*        I1     - PRIME IMPLICANT ROW IN FAULT TABLE        *
C*        IF1    - ARRAY OF CORRECT OUTPUTS                  *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS               *
C*        IN     - NUMBER OF INPUT VARIABLES                 *
C*************************************************************
       SUBROUTINE REDUCE
       COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
      1NUT(128,7),IV(14),S(128,1600)
       LOGICAL*1 S
       COMMON /CMAIN/K2,I1,J,IROW(128),ICOL(1600),II(128)
       INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,K2,I1,J,ICOL,
      1IROW,II
C
C      REMOVE FROM FAULT TABLE EACH COLUMN WITH AN X IN THE
C      PRIME IMPLICANT ROW
C
       K=0
       J1=K1
       DO 3 J3=1,K2
       IF(S(I1,J3)) GO TO 3
       K=K+1
       IF(K.EQ.J3) GO TO 3
       ICOL(K)=ICOL(J3)
       DO 4 J4=1,K1
     4 S(J4,K)=S(J4,J3)
     3 CONTINUE
C
C      REMOVE PRIME IMPLICANT ROW AND ANY OTHER ROW OR
C      COLUMN WHICH CONTAINS NO FAULTS
C
       J4=1
     8 K1=I1-1
       IF(K1.EQ.J1-1) GO TO 10
     2 K1=K1+1
       IF(K.EQ.0) RETURN
       IROW(K1)=IROW(K1+1)
       DO 5 J3=1,K
     5 S(K1,J3)=S(K1+1,J3)
       IF(K1.NE.J1-1) GO TO 2
C
```

```
C       REMOVE PRIME IMPLICANT FROM INPUT COMBINATIONS
C       AND CORRECT OUTPUT FROM OUTPUT ARRAY
C
        DO 6 J8=I1,K1
        DO 6 J2=1,IN
        IF1(J8)=IF1(J8+1)
    6   INPUT(J8,J2)=INPUT(J8+1,J2)
   10   IF(J4.EQ.0) GO TO 9
C
C       CALL SUBROUTINE RECNT TO RECOUNT THE NUMBER OF X'S
C       IN EACH ROW AND COLUMN OF FAULT TABLE
C
        CALL RECNT
        I1=1
        J4=0
    9   IF(I1.GE.K1) RETURN
        DO 7 J8=I1,K1
        IF(IROW(J8).NE.0) GO TO 7
        I1=J8
        J1=K1
        GO TO 8
    7   CONTINUE
        RETURN
        END
```

```
C      SUBROUTINE ROWS REMOVES DOMINATED ROWS FROM THE
C      FAULT TABLE
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        K1     - NUMBER OF ROWS IN FAULT TABLE             *
C*        I3     - STARTING POINT TO BEGIN CHECKING FOR      *
C*                        ROW DOMINATION                     *
C*        IROW   - ARRAY CONTAINING NUMBER OF FAULTS PER ROW *
C*        S      - FAULT TABLE                               *
C*        K      - NUMBER OF COLUMNS IN FAULT TABLE          *
C*        IF1    - ARRAY OF CORRECT OUTPUTS                  *
C*        INPUT  - ARRAY OF INPUT COMBINATIONS               *
C*        IN     - NUMBER OF INPUT VARIABLES                 *
C*************************************************************
       SUBROUTINE ROWS
       COMMON /AMAIN/INPUT(128,7),IF1(128),IF2(128),IN,K,K1,
      1NUT(128,7),IV(14),S(128,1600)
       LOGICAL*1 S
       COMMON /CMAIN/K2,I1,J,IROW(128),ICOL(1600),II(128)
       INTEGER*2 INPUT,IF1,IN,K,K1,NUT,IV,IF2,K2,I1,J,ICOL,
      1IROW,II
       I3=1
    7  J1=K1
       IF(I3.GE.K1) RETURN
       IF(K1.EQ.2) I3=1
C
C      COMPARE NUMBER OF X'S IN IROW(I) AND IROW(J)
C      IF IROW(I) = IROW(J), REMOVE IROW(J)
C
       DO 1 I=I3,K1
       IF(I.EQ.K1) RETURN
       I8=I+1
       DO 1 J=I8,K1
       IF(IROW(I).NE.IROW(J)) GO TO 4
       DO 2 I1=1,K
       IF((.NOT.S(J,I1).AND.S(I,I1)).OR.(S(J,I1).AND..NOT.
      1S(I,I1))) GO TO 1
    2  CONTINUE
       GO TO 12
    4  IF(IROW(I).LT.IROW(J)) GO TO 9
C
C      IF IROW(I) IS DOMINANT OVER IROW(J), REMOVE IROW(J)
C
       DO 10 J3=1,K
       IF(.NOT.S(J,J3)) GO TO 10
       IF(.NOT.S(I,J3)) GO TO 1
   10  CONTINUE
C
C      INITIALIZING VARIABLES TO REMOVE IROW(J)
C
```

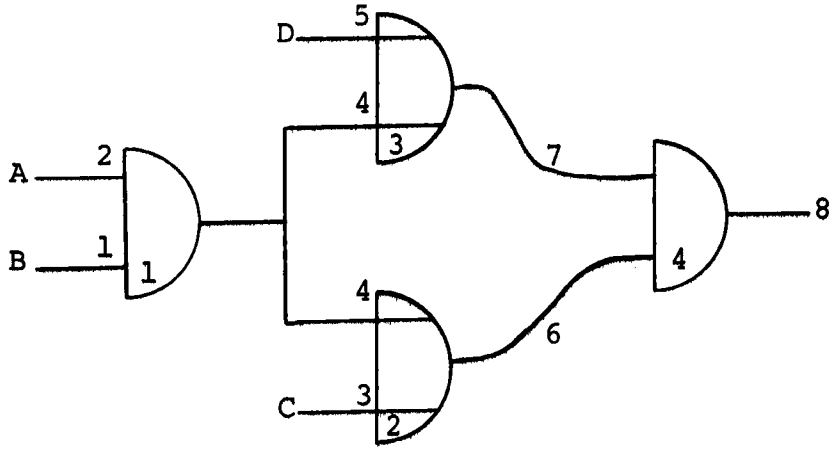
```
   12 K1=J-1
      I2=J
      I3=I
      GO TO 6
C
C     IF IROW(J) IS DOMINANT OVER IROW(I), REMOVE IROW(I)
C
    9 DO 11 J3=1,K
      IF(.NOT.S(I,J3)) GO TO 11
      IF(.NOT.S(J,J3)) GO TO 1
   11 CONTINUE
C
C     INITIALIZING VARIABLES TO REMOVE IROW(I)
C
      K1=I-1
      I2=I
      I3=I
      GO TO 6
    1 CONTINUE
      RETURN
    6 K1=K1+1
      IF(K1.NE.J1) GO TO 13
      I3=I3+1
      K1=K1-1
      IF(K1.EQ.1.AND.J1.EQ.2) RETURN
      GO TO 7
   13 IROW(K1)=IROW(K1+1)
      DO 5 J3=1,K
    5 S(K1,J3)=S(K1+1,J3)
      IF(K1.NE.J1-1) GO TO 6
C
C     REMOVE APPROPRIATE ROW FROM ARRAY OF CORRECT OUTPUTS
C     AND ARRAY OF INPUT COMBINATIONS
C
      DO 3 J3=I2,K1
      DO 3 I1=1,IN
      IF1(J3)=IF1(J3+1)
    3 INPUT(J3,I1)=INPUT(J3+1,I1)
      GO TO 7
      END
```

```
C      SUBROUTINE COMB COMPUTES THE COMBINATION OF N ITEMS
C      TAKEN IR AT A TIME
C
C*************************************************************
C*     VARIABLE AND ARRAY IDENTIFICATION                     *
C*        N      - NUMBER OF ROWS                            *
C*        IR     - ROWS TAKEN AT A TIME                      *
C*        FIRST  - LOGICAL VARIABLE    T-INITIAL OR NEW IR   *
C*                                     F-SAME IR             *
C*        I      - ARRAYS WHICH CONTAINS THE COMBINATIONS    *
C*************************************************************
       SUBROUTINE COMB(FIRST,N,IR,I)
       DIMENSION I(IR)
       INTEGER*2 N,I
       LOGICAL FIRST
       IF(FIRST) GO TO 1
       IF(I(IR).GE.N) GO TO 2
       I(IR)=I(IR)+1
       RETURN
     1 DO 3 J=1,IR
     3 I(J)=J
       FIRST=.FALSE.
       RETURN
     2 J=IR
     7 IF(I(J-1).LT.(N-IR+J-1)) GO TO 4
       IF(J.EQ.2) GO TO 6
       J=J-1
       GO TO 7
     4 I(J-1)=I(J-1)+1
       DO 5 IS=J,IR
     5 I(IS)=I(J-1)+IS-(J-1)
       RETURN
     6 FIRST=.TRUE.
       RETURN
       END
```

APPENDIX B

SAMPLE INPUTS AND OUTPUTS

CASE 1 - PAGE 18 OF THIS THESIS



INPUT DATA

AND	2	2	B	A
OR	2	1	C	1
OR	2	1	1	D
AND	2	1	2	3

PROGRAM OUTPUT

THE COMBINATIONAL CIRCUIT

GATE NUMBER	GATE TYPE	NO OF INPUTS	1ST INPUT	2ND INPUT	3RD .... 8TH INPUT
1	AND	2	B	A	
2	OR	2	C	1	
3	OR	2	1	D	
4(OUTPUT)	AND	2	2	3	

THE FAULT TABLE

INPUTS	CORRECT OUTPUT	POSSIBLE S-A-0 FAULTS								POSSIBLE S-A-1 FAULTS							
		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
ABCD	F	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0000	0												X				X
0001	0											X	X		X		X
0010	0												X	X		X	X
0011	1			X		X	X	X	X								
0100	0										X		X				X
0101	0										X	X	X		X		X
0110	0										X		X	X		X	X
0111	1			X		X	X	X	X								
1000	0									X			X				X
1001	0									X		X	X		X		X
1010	0									X			X	X		X	X
1011	1			X		X	X	X	X								
1100	1	X	X		X		X	X	X								
1101	1	X	X		X		X	X	X								
1110	1	X	X		X		X	X	X								
1111	1						X	X	X								

A MINIMAL FAULT-DETECTION EXPERIMENT
CONSISTS OF THE FOLLOWING TESTS

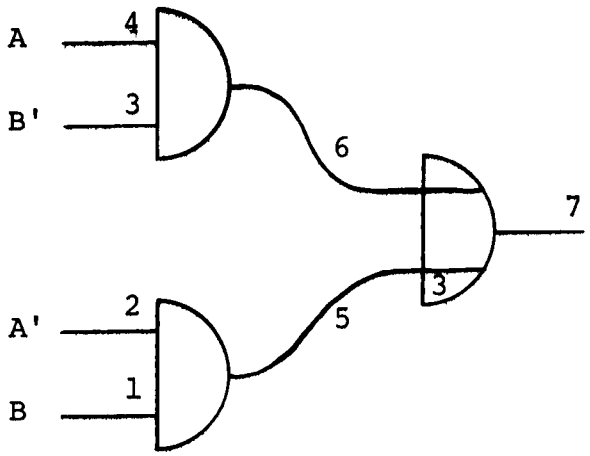
1) APPLY 1100: IF THE OUTPUT IS F = 0, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

2) APPLY 0011: IF THE OUTPUT IS F = 0, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

3) APPLY 0101: IF THE OUTPUT IS F = 1, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

4) APPLY 1010: IF F = 1, CIRCUIT IS FAULTY; OTHERWISE, CIRCUIT OPERATES CORRECTLY

CASE 2 - PAGE 24 OF THIS THESIS



INPUT DATA

```
AND     2    1    B   A'
AND     2    1   B'    A
OR      2    1    1    2
```

PROGRAM OUTPUT

THE COMBINATIONAL CIRCUIT

GATE NUMBER	GATE TYPE	NO OF INPUTS	1ST INPUT	2ND INPUT	3RD .... 8TH INPUT
1	AND	2	B	A'	
2	AND	2	B'	A	
3(OUTPUT)	OR	2	1	2	

THE FAULT TABLE

INPUTS	*CORRECT OUTPUT	*	POSSIBLE S-A-0 FAULTS							*	POSSIBLE S-A-1 FAULTS						
			*1	2	3	4	5	6	7		*1	2	3	4	5	6	7
AB	*	F	*0	0	0	0	0	0	0		*1	1	1	1	1	1	1
00		0									X			X	X	X	X
01		1	X	X			X		X								
10		1			X	X		X	X								
11		0										X	X		X	X	X

A MINIMAL FAULT-DETECTION EXPERIMENT CONSISTS OF THE FOLLOWING TESTS

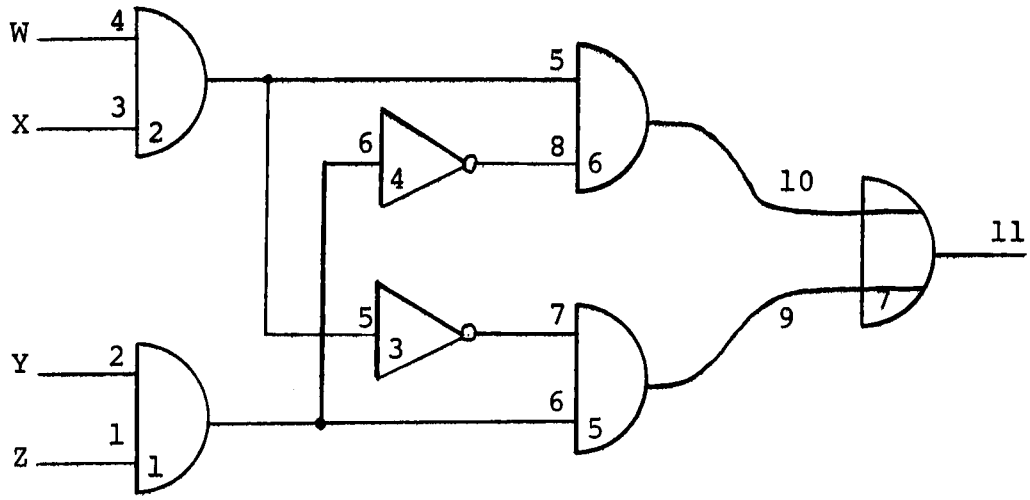
1) APPLY 01: IF THE OUTPUT IS F = 0, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

2) APPLY 10: IF THE OUTPUT IS F = 1, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

3) APPLY 00: IF THE OUTPUT IS F = 1, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

4) APPLY 11: IF F = 0, CIRCUIT IS FAULTY; OTHERWISE, CIRCUIT OPERATES CORRECTLY

CASE 3 - PAGE 33 OF THIS THESIS



INPUT DATA

```
AND     2   2   Z   Y
AND     2   2   X   W
NOT     1   1   2
NOT     1   1   1
AND     2   1   1   3
AND     2   1   4   2
OR      2   1   5   6
```

PROGRAM OUTPUT

THE COMBINATIONAL CIRCUIT

GATE NUMBER	GATE TYPE	NO OF INPUTS	1ST INPUT	2ND INPUT	3RD .... 8TH INPUT
1	AND	2	Z	Y	
2	AND	2	X	W	
3	NOT	1	2		
4	NOT	1	1		
5	AND	2	1	3	
6	AND	2	4	2	
7(OUTPUT)	OR	2	5	6	

THE FAULT TABLE

INPUTS	CORR OUTPT	POSSIBLE S-A-0 FAULTS											POSSIBLE S-A-1 FAULTS									
		1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9	10	11
WXYZ	F	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0000	0																X	X			X	X	X
0001	0													X			X	X			X	X	X
0010	0												X				X	X			X	X	X
0011	1	X	X			X	X		X			X					X						
0100	0															X	X	X			X	X	X
0101	0													X		X	X	X			X	X	X
0110	0												X			X	X	X			X	X	X
0111	1	X	X			X	X		X			X				X	X						
1000	0														X		X	X			X	X	X
1001	0													X	X		X	X			X	X	X
1010	0												X		X		X	X			X	X	X
1011	1	X	X			X	X		X			X			X		X						
1100	1			X	X	X			X		X		X					X					
1101	1			X	X	X			X		X		X		X			X					
1110	1			X	X	X			X		X		X	X				X					
1111	0	X	X	X	X	X	X												X	X	X	X	X

A MINIMAL FAULT-DETECTION EXPERIMENT CONSISTS OF THE FOLLOWING TESTS

1) APPLY 1111: IF THE OUTPUT IS F = 1, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

2) APPLY 0101: IF THE OUTPUT IS F = 1, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

3) APPLY 1011: IF THE OUTPUT IS F = 0, CIRCUIT IS FAULTY AND THE EXPERIMENT MAY BE STOPPED; OTHERWISE, APPLY NEXT TEST

4) APPLY 1110: IF F = 0, CIRCUIT IS FAULTY; OTHERWISE, CIRCUIT OPERATES CORRECTLY

VITA

Thomas Earl Akin was born in Pikeville, Tennessee, on June 2, 1947. He attended elementary school in that city and was graduated as valedictorian from Bledsoe County High School in May of 1965. He received a Bachelor of Science degree in Mathematics from Lambuth College in June 1969.

In 1971, after two years of military service, he began study toward a Master's degree at the University of Tennessee. In May 1972, he accepted a teaching assistantship and taught commercial-oriented programming courses at the undergraduate level. In June, 1973, he received a Master of Science degree with a major in Computer Science. He is a member of the Association for Computing Machinery.