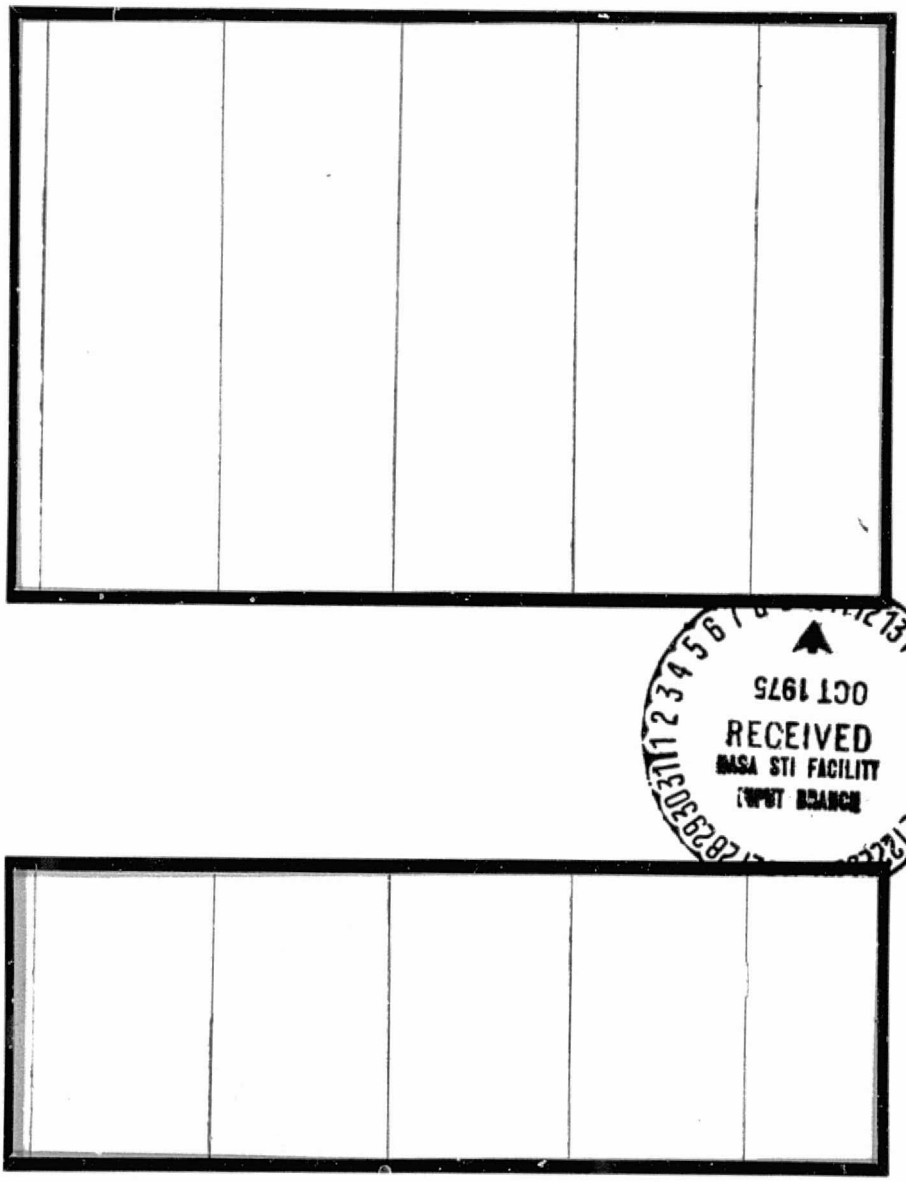
ELECTRICAL

ENGINEERING



ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA

FINAL TECHNICAL REPORT

DEVELOPMENT OF AUTOMATED TEST PROCEDURES AND TECHNIQUES FOR LSI CIRCUITS

Prepared by

B. D. Carroll
Electrical Engineering Department
Auburn University
Auburn, Alabama 36830

Contract NAS8-31190

George C. Marshall Space Flight Center
National Aeronautics and Space Administration
Marshall Space Flight Center, Alabama 35812

September 4, 1975

TABLE OF CONTENTS

ABSTRACT

Testing of large scale integrated (LSI) logic circuits is considered from the point of view of automatic test pattern generation. A system for automatic test pattern generation is described. A test generation algorithm is presented that can be applied to both combinational and sequential logic circuits. Also included is a programmed implementation of the algorithm and sample results from the program. Recommendations for continued study are also discussed.

List of Figures

List of Tables

CHAPTER 1 - INTRODUCTION

Logic circuit testing as viewed in this report is the process of examining a logic circuit to determine whether or not the circuit correctly performs the desired logic function. It will be assumed that a correct logic design has been accomplished and that abnormal functions are produced by failures called faults introduced during the manufacturing process or at some time later in the life of the circuit. Logic testing is often performed as one of the final stages in the manufacture of a logic circuit. Also, logic testing is frequently used as an acceptance test by a purchaser of logic circuits. Logic testing is also required during the checkout and maintenance of logic circuit assemblies and logic systems. This report will be oriented toward the automated testing of integrated circuits as they come off a manufacturing line.

The logic testing process is accomplished by applying a sequence of input patterns to a powered circuit and observing the corresponding sequence of responses.† A circuit is assumed to be fault-free if all the observed responses are correct. Incorrect responses, on the other hand, signal a circuit containing some fault condition. Observation of the complete response sequence produced by a circuit containing a fault may contain enough information to identify precisely the fault condition present. However, the identification of the particular fault present in

†It is assumed that circuits are allowed to reach a stable state before each input pattern is applied.

a faulty logic circuit is usually not important for integrated circuit testing since repair is usually not possible.

Fault detection testing is a term often used to identify the process of testing a circuit to determine whether or not the circuit contains a fault. Fault location testing describes the process of identifying the fault present. A term that combines the meaning of the above is fault diagnosis testing.

The input patterns that are applied during the testing process are referred to as test patterns. Selection of the test patterns sequence is one of the most difficult aspects of the testing problem. The remaining sections of this report will be concerned with test pattern selection problem - often called test pattern generation (TPG).

The remainder of this chapter is devoted to a brief survey of previous work on the sequential circuit TPG problem and on a discussion of a practical TPG system. Chapter 2 contains a presentation of a logic model and a circuit analysis procedure that can be used to generate test patterns. Two methods of using the analysis procedure to generate test patterns are described in Chapter 3. A deficiency of the analysis procedure concerned with starting state specification is considered in Chapter 4. Conclusions and recommendations for future work are given in the final chapter. A brief bibliography is also included. Description of a programmed implementation of the analysis procedure is contained in an appendix.

PREVIOUS WORK

Many papers and reports have been published that treat various aspects of test pattern generation for sequential logic circuits. However, much work remains to be done in the development of practical automatic test pattern generation procedures.

Bouricius, et.al [1] have described an extension of the d-Algorithm that can be applied to asynchronous circuits. The use of the Boolean difference for sequential circuit test pattern generation has been considered by Hsiao and Chia [2]. A random technique for test pattern generation was described by Breuer [3]. Use of random techniques in a specific test application is presented in a paper by Agrawal and Agrawal [4]. Testing for intermittent fault detection has been treated by Breuer[5]. Chappell [6]has described a test pattern generation procedure for sequential circuits that was developed at Bell Laboratories. The approach of[6] will be considered in much detail in the remaining chapters of this report.

TEST PATTERN GENERATION SYSTEM

A practical approach to test pattern generation is described in this section. The following features are desirable if not necessary in any useful test pattern generation system.

1. Accommodate both combinational and sequential logic circuits.
2. Handle circuits equivalent in complexity to five thousand logic gates.
3. Produce test pattern sequences that detect greater than ninety-five per cent of all single stuck-at faults.
4. Produce practical length test sequences.
5. Avoid test sequences that induce races in asynchronous circuits.
6. Are cost effective in terms of computer costs and manhours.
7. Allow for easy upgrading of capability.

The system shown in Figure 1.1 is designed to satisfy the above criteria. Each component of the system is described in detail in the paragraphs that follow. In general, the approach is to obtain a "first pass" test sequence using a fast TPG method then to evaluate the sequence using fault simulation and finally to update the test sequence if needed with the aid of a more sophisticated TPG procedure or manually.

Primary Test Pattern Generator

This step in TPG should produce a test sequence that will detect approximately eighty-five to ninety per cent of the faults represented by the fault model. This process should not require that faults be

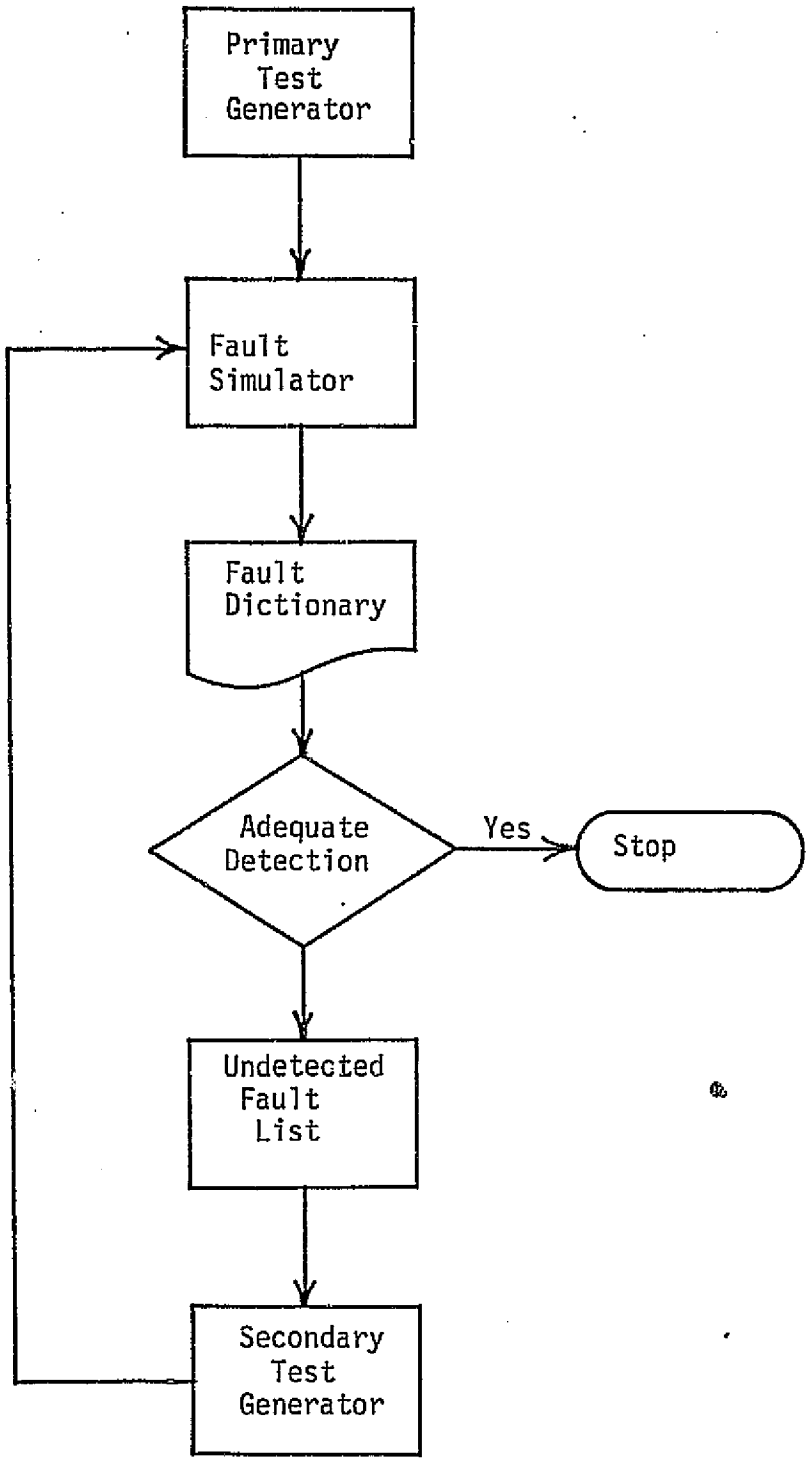


Figure 1,1. TEST PATTERN GENERATION SYSTEM FLOWCHART

separately identified during generation of tests and should be fast in terms of computer time per fault detected. The procedure described in C h a p t e r 3 can be used in this mode of operation. Random methods for generating test sequences may also prove useful here although little work has been reported on such approaches at this time.

Fault Simulation

The second step of the process is for the evaluation of the test sequence produced above. Fault simulation is the most cost effective means for providing this evaluation. The effectiveness and validity of this step is determined by the accuracy of the simulator used for the analysis. However, as simualtion accuracy is improved, the cost of simulation is increased. A time-based event-driven simulator provides the best accuracy, but a unit delay simualtor may be useful for some circuits. Parallel or deductive fault simulators are needed for fault simulation of practical sized circuits.

Secondary Test Pattern Generator

The function of this step of the TPG process is to determine test sequences for detecting those faults that would not be detected by the sequence produced by the primary test pattern generator. Faults for which tests are desired should be specified to the generator individually. The test generator should be based on a circuit model close to that used in logic simulators. The procedure given in C h a p t e r 3 can also be used for the secondary TPG.

Flexibility

Flexibility is provided in the system described above by the independence of the major components. Therefore a system can be assembled with available test pattern generators and fault simulators. Components can be replaced with improved versions when they become available. However, development of a common data base for all programs in the system is desirable.

CHAPTER 2 - BACKGROUND

This chapter is devoted to a discussion of the logic circuit model and analysis procedure that forms the basis of the test pattern generation methods described in Chapter 3. The model and procedure given here have been adopted from the approach presented by Chappell [6]. However, major changes have been made in the race detection and prevention portion of the analysis procedure. A program of the procedure is described in the appendices.

Logic circuits will be assumed to be interconnections of NAND gates with unit delay assumed as the propagation time of each gate. A generalization to handle other types of logic gates can be accomplished in a straightforward manner as given in [6].

LOGIC CIRCUIT MODEL

Consider the NAND gate shown in Figure 2.1. Each input and output of the gate is represented by an ordered pair of Boolean variables. The meaning of each possible combination of a variable pair is given in Table 2.1. A NAND gate truth table in terms of variable pairs is presented in Table 2.2. From the truth table, the following pair of Boolean equations can be obtained to represent a NAND gate.

$$C^1 = A^0 + B^0$$
$$C^0 = A^1 B^1$$

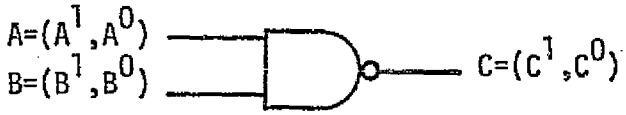


Figure 2.1. TWO-INPUT NAND GATE

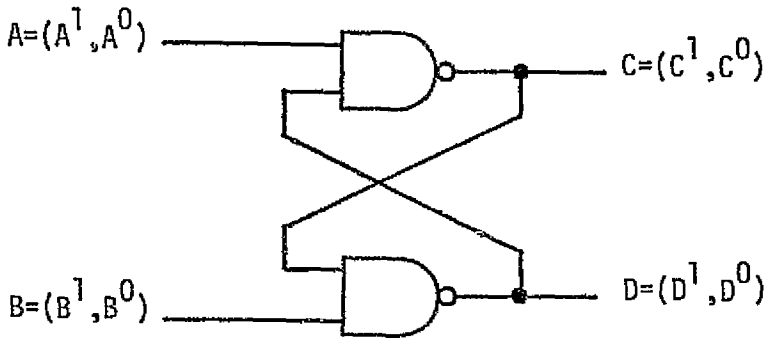


Figure 2.2. CROSS-COUPLED NAND GATE PAIR

TABLE 2.1

Variable Pair Combination of Values

VARIABLE PAIR	INTERPRETATION
(0,0)	Unknown Logical Value
(0,1)	Logical 0
(1,0)	Logical 1
(1,1)	Undefined - not allowed

TABLE 2.2

NAND Gate Truth Table

(A^1,A^0)	(B^1,B^0)	(C^1,C^0)
(0,1)	(0,1)	(1,0)
(0,1)	(1,0)	(1,0)
(1,0)	(0,1)	(1,0)
(1,0)	(1,0)	(0,1)

An important feature of the variable pair representation is that a means exists to describe an unknown logical value as well as the usual logical 1 and logical 0 values. However, the individaual variables are related by the standard Boolean relationships OR and AND and therefore can be easily manipulated. The Boolean complement is not utilized.

Sequential logic circuits are represented in a similar manner to that described above for a NAND gate. The set of equations below represent the cross-coupled NAND gates shown in Figure 2.2.

$$C^1 = A^0 + D^0$$
$$C^0 = A^1 D^1$$
$$D^1 = B^0 + C^0$$
$$D^0 = B^1 C^1$$

CIRCUIT ANALYSIS PROCEDURE

The circuit model presented above will now be used as the basis of a procedure for developing a set of time dependent Boolean equations that describe all ways of placing the logic signals in a circuit in each logical state at a given time. An illustration of the procedure will be given following a discussion of the timing relationships of the logical variables.

Two time parameters are used in the timing model. One parameter called input time and denoted by t describes the times at which circuit input signals are changed. The other time parameter is called ripple time and is designated by the symbol r. Ripple time represents the propagation in time of signals through a circuit due to gate delays and is incremented in unit steps consistent with the unit delay assumption. Input time is

incremented only after a circuit has reached a stable condition. Hence circuit inputs are functions of input times only, but gate outputs are functions of both input time and ripple time. The following time dependent Boolean equations result for the cross-coupled NAND gates of Figure 2.2.

$$C^1(t,r) = A^0(t) + D^0(t, r-1)$$
$$C^0(t,r) = A^1(t)\, D^1(t, r-1)$$
$$D^1(t,r) = B^0(t) + C^0(t, r-1)$$
$$D^0(t,r) = B^1(t)\, C^1(t, r-1)$$

A set of time dependent equations can be developed from the basic set of equations that extend from t=0 to a specified time. Table 2.3 summarizes this process for the cross-coupled NAND gates. It should be observed that each variable was initially assumed to be unknown in this example. Also notice that input time t=2 was established only after the same equation set was obtained for two consecutive ripple times which indicates that a stable condition was reached.

Interpretation of the equations in Table 2.3 is in order. Consider the equations at t=1, r=3. These equations represent all ways of controlling the outputs of the circuits in one input time step assuming an unknown starting state for the circuit outputs. More specifically, $C^1(1,3) = A^0(1)$ implies that output C can be forced to a logical 1 state by applying a logical 0 to input A. Similarly, $C^0(1,3) = A^1(1)B^0(1)$ states that C can be forced to 0 by applying 1 to input A and 0 to input B. Discussion of the equations for t=2 will be deferred until later.

TABLE 2.3

Equation Development for Cross-Coupled NAND Gate Pair

t	r	$C^1(t,r)$	$C^0(t,r)$	$D^1(t,r)$	$D^0(t,r)$
0	0	0	0	0	0
1	1	$A^0(1)$	0	$B^0(1)$	0
	2	$A^0(1)$	$A^1(1)B^0(1)$	$B^0(1)$	$A^0(1)B^1(1)$
	3	$A^0(1)$	$A^1(1)B^0(1)$	$B^0(1)$	$A^0(1)B^1(1)$
2	4	$A^0(2) + A^0(1)B^1(1)$	$A^1(2)B^0(1)$	$B^0(2) + A^1(1)B^0(1)$	$A^0(1)B^1(2)$
	5	$A^0(2) + A^0(1)B^1(2)$	$A^1(2)B^0(2) + A^1(2)A^1(1)B^0(1)$	$B^0(2) + A^1(2)B^0(1)$	$A^0(2)B^1(2) + A^0(1)B^1(2)B^1(1)$
	6	$A^0(2) + A^0(1)B^1(2)B^1(1)$	$A^1(2)B^0(2) + A^1(2)B^0(1)$	$B^0(2) + A^1(2)A^1(1)B^0(1)$	$A^0(2)B^1(2) + A^0(1)B^1(2)$
	7	$A^0(2) + A^0(1)B^1(2)$	$A^1(2)B^0(2) + A^1(2)A^1(1)B^0(1)$	$B^0(2) + A^1(2)B^0(1)$	$A^0(2)B^1(2) + A^0(1)B^1(2)B^1(1)$

Special steps may be necessary in order that stability be reached in sequential circuits. As can be seen in Table 2.3 the equation set starts repeating at r=7. This oscillation is caused by the feedback present in the circuit and by the fact that the equation development process permitted unrestricted input changes. It is well known that a 00 to 11 input change causes a race condition in cross-coupled NAND gates. Race conditions are manifested in the circuit model by oscillating equation sets. However, race conditions can be avoided as discussed in the next section.

RACE ANALYSIS

It is shown in [6] that a race condition is first indicated when both the equations for C^0 and D^1 or D^0 and C^1 change from the previous ripple time. When one of these conditions is detected, C^0 and D^0 can be systematically modified so that input changes are restricted to prevent a race condition from occuring. Race analysis consists of the detection and the prevention of race conditions.

For the circuit in Figure 2.2, Chappell [6] has shown that if the equation for C^0 does not change from the previous ripple time then no race will occur and it is not necessary to check for changes in other equations. However, the author has found that race conditions can be overlooked if this rule is applied to the circuit in Figure 2.3. Hence, the race detection procedure adopted here is to check C^0 and D^1 plus D^0 and C^1 for changes in all cross-coupled NAND gates in a circuit under analysis.

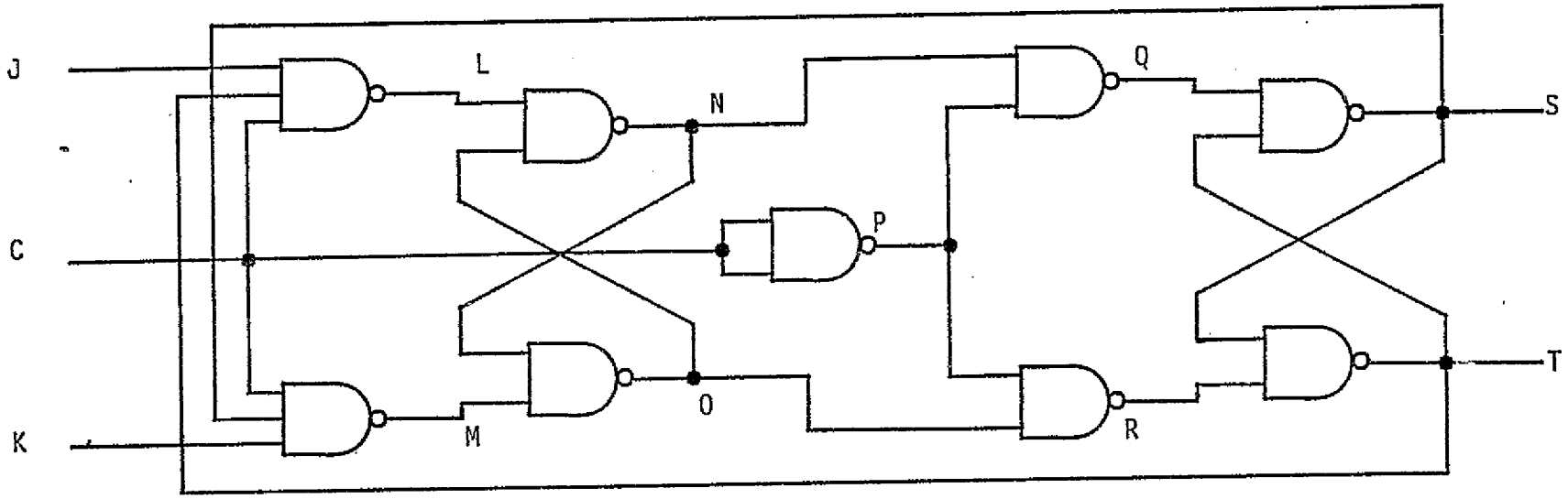


Figure 2.3. JK FLIP-FLOP LOGIC CIRCUIT

When a race condition is detected in a pair of cross-coupled NAND gates, equations C^0 or D^1 and D^0 or C^1 are modified in such a manner to eliminate input sequences that would cause the race to occur. The equation modification rules adopted here are given in Table 2.4. Rules I.A and II.A are the same as those given in [6]. An explanation of the rules in Table 2.4 will now be given.

Consider the possible combinations of outputs of the cross-coupled NAND gates of Figure 2.1. The output combinations that do not indicate a race condition are shown in Figure 2.4a. Possible race conditions are indicated by those combinations given in Figure 2.4b.

As can be seen, the following functional relationships hold when no race condition is indicated.

$$C^1 \supseteq D^0 \tag{1a}$$

$$D^1 \supseteq C^0 \tag{1b}$$

On the other hand, the functional relationships below are true for the conditions that indicate a possible race.

$$C^1 \subseteq D^0 \tag{2a}$$

$$D^1 \subseteq C^0 \tag{2b}$$

Hence, it can be concluded from (1) and (2) that no race can occur if the equations that represent the outputs of cross-coupled NAND gates are forced to satisfy the relationships of (1) when at least one equation of each pair is non-zero. The rules given in Table 2.4 modify the equations so the desired relationships are satisfied. Table 2.5 shows the application of these rules to the analysis of the cross-coupled NAND gate circuit.

TABLE 2.4

Race Prevention Rules

I. Modification of C^0 or D^1.

A. If $C^0(t,r) \neq 0$ and $D^1(t,r) \neq 0$

Then $C^0(t,r) = C^0(t,r) \cdot D^1(t,r)$.

B. If $C^0(t,r) \neq 0$ and $D^1(t,r) = 0$,

Then $D^1(t,r) = C^0(t,r)$.

C. If $C^0(t,r) = 0$, then no change.

II. Modification of D^0 and C^1.

A. If $D^0(t,r) \neq 0$ and $C^1(t,r) \neq 0$,

Then $D^0(t,r) = D^0(t,r) \cdot C^1(t,r)$.

B. If $D^0(t,r) \neq 0$ and $C^1(t,r) = 0$, then $C^1(t,r) = D^0(t,r)$.

C. If $D^0(t,r) = 0$, then no change.

Logical Value		Variable-Pair Representation			
C	D	C^1	C^0	D^1	D^0
X	X	0	0	0	0
X	1	0	0	1	0
0	1	0	1	1	0
1	X	1	0	0	0
1	0	1	0	0	1
1	1	1	0	1	0

(a) Race-Free Conditions

Logical Value		Variable-Pair Representation			
C	D	C^1	C^0	D^1	D^0
X	0	0	0	0	1
0	X	0	1	0	0
0	0	0	1	0	1

(b) Possible Race Conditions

Figure 2.4 Analysis of Cross-Coupled NAND Gate Outputs

TABLE 2.5

Equation Development with Race Analysis

t	r	$C^1(t,r)$	$C^0(t,r)$	$D^1(t,r)$	$D^0(t,r)$
0	0	0	0	0	0
1	1	$A^0(1)$	0	$B^0(1)$	0
	2	$A^0(1)$	$A^1(1)B^0(1)$	$B^0(1)$	$A^0(1)B^1(1)$
	3	$A^0(1)$	$A^1(1)B^0(1)$	$B^0(1)$	$A^0(1)B^1(1)$
2	4	$A^0(2)+A^0(1)B^1(1)$	$A^1(2)B^0(1)$	$B^0(2)+A^1(1)B^0(1)$	$A^0(1)B^1(2)$
	4R		$A^1(2)b^0(2)B^0(1)+$ $A^1(2)A^1(1)B^0(1)$		$A^0(2)A^0(1)B^1(2)+A^0(1)B^1(2)B^1(1)$
	5	$A^0(2)+A^0(1)B^1(2)B^1(1)$	$A^1(2)B^0(2)+A^1(1)B^0(1)$	$B^0(2)+A^1(2)A^1(1)B^0(1)$	$A^0(2)B^1(2)+A^0(1)B^1(2)B^1(1)$
	5R		$A^1(2)B^0(2)+A^1(1)B^0(1)$		$A^0(2)B^1(2)+A^0(1)B^1(2)B^1(1)$
	6	$A^0(2)+A^0(1)B^0(2)B^1(1)$	$A^1(2)B^0(2)+A^1(1)B^0(1)$	$B^0(2)+A^1(2)A^1(1)B^0(1)$	$A^0(2)B^1(2)+A^0(1)B^1(2)B^1(1)$

Now consider the interpretation of the equations in Table 2.5 for t=2, r=6. These equations represent all race-free ways of controlling the circuit outputs in two input time steps. In particular, the equation

$$C^1(2,6) = A^0(2) + A^0(1)B^1(2)B^1(1)$$

indicates two ways of placing output C in the logical 1 state after two input time steps. First, input A can be set to 0 at t=2. Second, input A can be set to 0 and input B set to 1 at t=1 with B held at 1 for t=2. A is a don't care condition for t=2, in the second case. Similar meanings follow for the remaining equations.

OSCILLATION ANALYSIS

Equation oscillation may occur during the circuit analysis procedure even if race conditions are prevented in cross-coupled NAND gates. This non-race oscillation is caused by global feedback paths that lead to the possibility of closed conduction paths that contain an odd number of logic signal inversions and an odd number of unit delays.

No immediate means of predicting a non-race oscillation condition has been developed. However, such oscillations can be handled by setting an upper limit on the number of ripple times allowed for each input time step. An oscillation would be assumed to exist if the equation sets do not stabilize before the ripple time exceeds the established limit.

Ripple time limit should be set as a function of the circuit under analysis and is probably related to the number of inversions in the longest closed feedback path. Further study is needed to establish a precise means for setting this limit.

Chappell [6] has suggested applying the rule given by the equations below to oscillating equations such as F^1 or F^0 in an attempt to halt the oscillation process.

$$F^1(t,r) = F^1(t,r) \cdot F^1(t, r-1) \tag{3a}$$

$$F^0(t,r) = F^0(t,r) \cdot F^0(t, r-1) \tag{3b}$$

DETAILED PROCEDURE

Let $I_1, \ldots, I_n$ correspond to the primary inputs of a logic circuit, and let $J_{n+1}, \ldots, J_m$ correspond to the logic gate† outputs of the circuit. Let (I_i^1, I_i^2) and (J_j^1, J_j^0) represent the logic value of circuit input line i and gate output line j, respectively. Then for each j, $n + 1 \leq j \leq m$, the following pair of Boolean equations follow.

$$J_j^1(t,r) = \sum_{k \epsilon K_j} I_k^0(t) + \sum_{\ell \epsilon L_j} J_\ell^0(t, r-1) \tag{4a}$$

$$J_j^0(t,r) = \prod_{k \epsilon K_j} I_k^1(t) \cdot \prod_{\ell \epsilon L_j} J_\ell^1(t, r-1) \tag{4b}$$

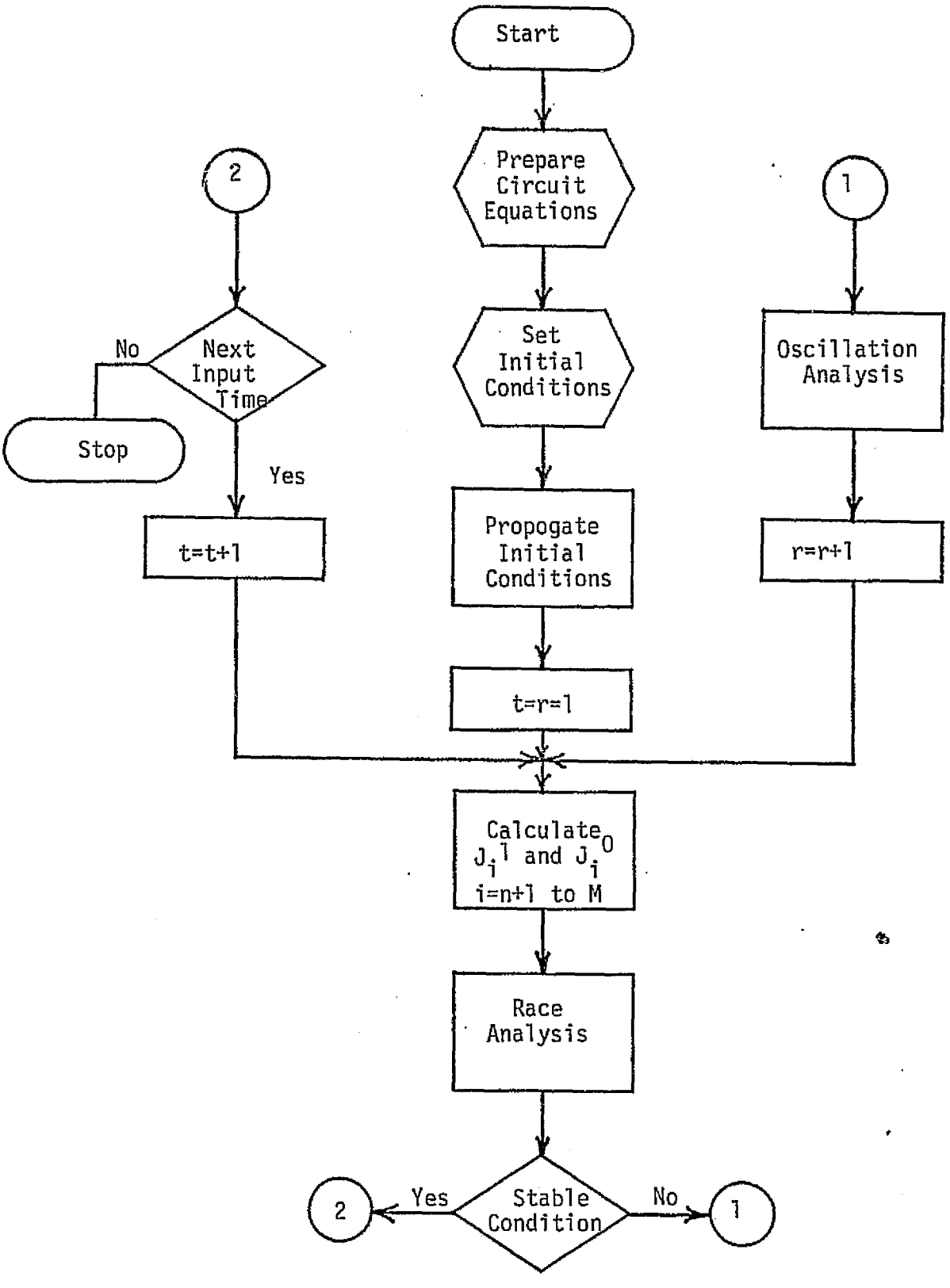
where K_j = {Primary inputs that are inputs of gate j}

and L_j = {Gate outputs that are inputs of gate j}

The circuit analysis procedure illustrated previously is flowcharted in Figure 2.5 in terms of the notation used in (4). Also, cross-coupled NAND gate outputs are denoted F and G in the race analysis procedure flowchart.

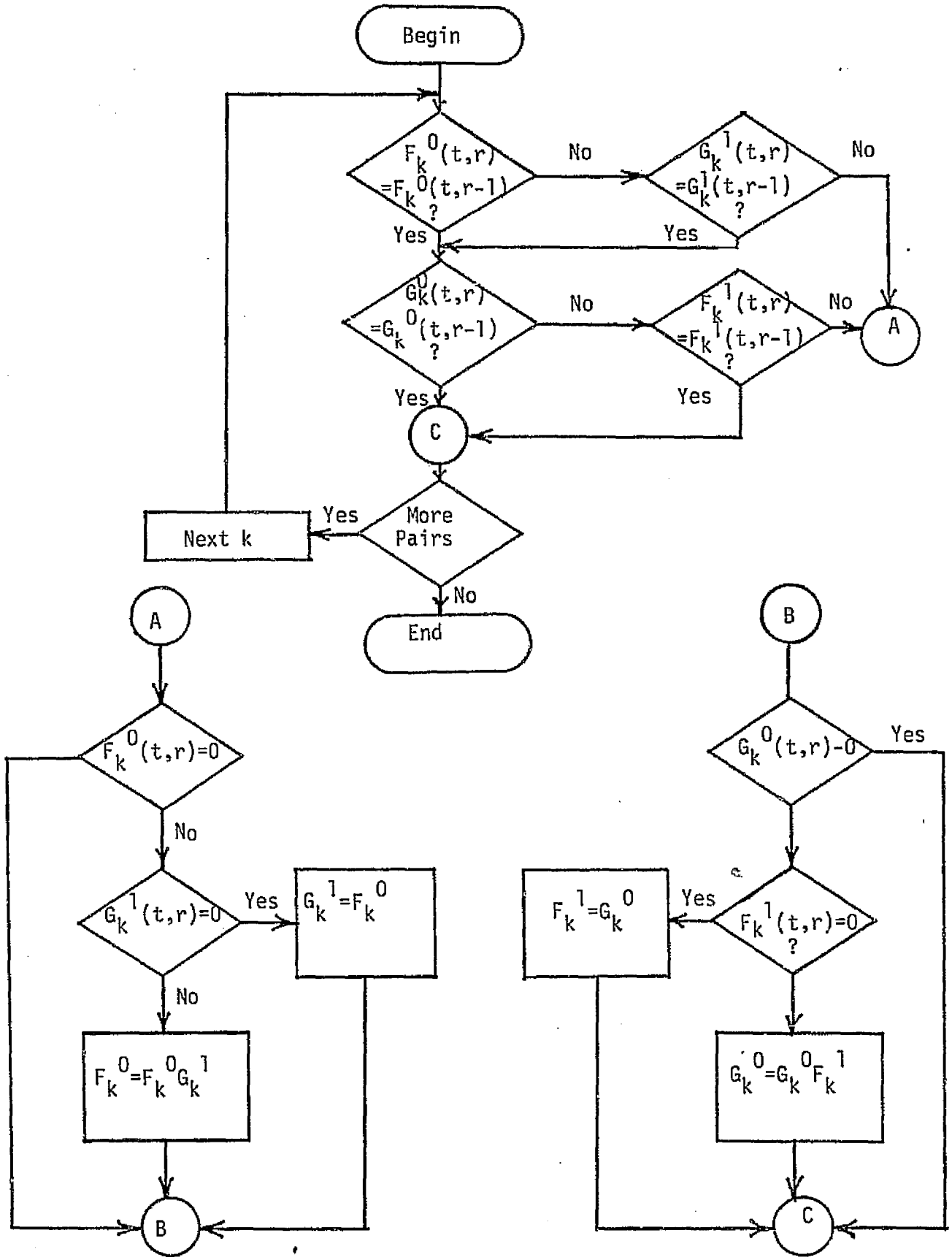
---

†NAND gates are assumed.



(a) Main Procedure

Figure 2 5 Circuit Analysis Procedure Flowchart



(b) Race Analysis Procedure

Figure 2.5. (con't) 23

CHAPTER 3 - TEST PATTERN GENERATION

The analysis procedure described in Chapter 2 can be used as the basis of two approaches for test pattern generation. Both approaches are taken from [6].

FAULT INSERTION

Faults are inserted in a logic circuit by modifying the equations that represent the faulty gate. An ordered pair of Boolean variables will also be used to represent faults. For example, let α correspond to some fault. The variable pair (α^1, α^0) will represent the fault where (1, 0) means the fault is present and (0, 1) means the fault is not present. This fault representation will be utilized in describing a circuit with a fault.

Any single stuck-at fault in an all NAND gate logic circuit is equivalent to some gate input stuck-at-1, some gate output stuck-at-0, or some gate output stuck-at-1. Let α_1 represent input A stuck-at-1 for the gate in Figure 3.1, and let γ_0 and γ_1 represent the output C stuck-at-0 and stuck-at-1, respectively. Table 3.1 summarizes the resulting equations for the fault free NAND gate and for each of the faults described above.

Let β_1 represent a stuck-at-1 fault on the B input of the corss-coupled NAND gates shown in Figure 3.2. The faulty circuit is described by the following basic set of equations.

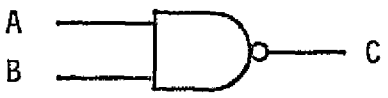


Figure 3.1. Two-Input NAND Gate with Fault

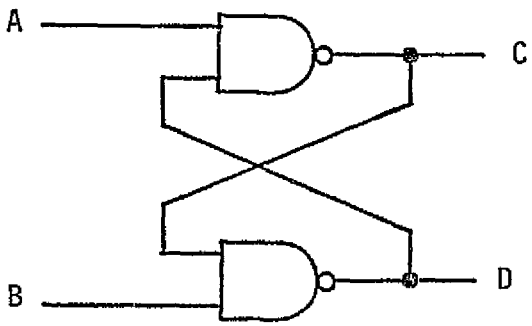


Figure 3.2. Cross-Coupled NAND Gate Pair with Fault

TABLE 3.1

Fault Insertion in Two-Input NAND Gate

Fault Free	A Stuck-at-1	C Stuck-at-0	C Stuck-at-1
$C^1 = A^0 + B^0$	$C^1 = A^0 \alpha_1^0 + B^0$	$C^1 = A^0 \gamma_0^0 + B^0 \gamma_0^0$	$C^1 = A^0 + B^0 + \gamma_1^1$
$C^0 = A^1 B^1$	$C^0 = \alpha_1^1 B^1 + A^1 B^1$	$C^0 = A^1 B^1 + \gamma_0^1$	$C^0 = A^1 B^1 \gamma_1^0$

$$C^1(t,r) = A^0(t) + D^0(t, r-1)$$
$$C^0(t,r) = A^1(t)D^1(t, r-1)$$
$$D^1(t,r) = \beta_1^0 B^0(t) + C^0(t, r-1)$$
$$D^0(t,r) = \beta_1^1 C^1(t, r-1) + B^1(t)C^1(t, r-1)$$

Propagation of the above equations is shown in Table 3.2 for two input time steps. It should be emphasized that race and oscillation analysis must be performed for the fault case in the same manner as for the fault-free case.

DETERMINATION OF TEST SEQUENCES

Let F represent an output of a logic circuit that has had fault ϕ inserted. The equation pair describing F in time can be written as follows.

$$F^1 = U + V\phi^1 + W\phi^0$$
$$F^0 = X + Y\phi^1 + Z\phi^0$$

where U, V, W, X, Y, and Z are Boolean expressions. All fault detection test sequences of length t for ϕ are given by the following test function.

$$F_\phi^t = VZ + WY \qquad (5)$$

For an example consider fault β_1 in the cross-coupled NAND gates. The results will be shown for only output C even though similar results can be obtained for output D.

At t = 1:

$$C^1(1) = B^0(1)\beta_1^0$$
$$C^0(1) = A^0(1)B^1(1) + A^0(1)\beta_1^1$$

TABLE 3.2

Equation Development with Inserted Fault

t	r	$C^1(t,r)$	$C^0(t,r)$	$D^1(t,r)$	$D^0(t,r)$
0	0	0	0	0	0
1	1	$A^0(1)$	0	$B_1^0B^0(1)$	0
	2	$A^0(1)$	$A^1(1)B_1^0B^0(1)$	$B_1^0B^0(1)$	$B_1^1A^0(1)+B^1(1)A^0(1)$
	3	$A^0(1)$	$A^1(1)B_1^0B^0(1)$	$B_1^0B^0(1)$	$B_1^1A^0(1)+B^1(1)A^0(1)$
2	4	$A^0(2)+B_1^1A^0(1)+B^1(1)A^0(1)$	$A^1(2)B_1^0B^0(1)$	$B_1^0B^0(2)+A^1(1)B_1^0B^0(1)$	$B_1^1A^0(1)+B^1(2)A^0(1)$
	4R		$A^1(2)B_1^0B^0(1)B^0(2)$ + $A^1(2)A^1(1)B_1^0B^0(1)$		$B_1^1A^0(1)+A^0(2)B^1(2)A^0(1)$ + $B^1(2)B^1(1)A^0(1)$
	5	$A^0(2)+B^1(2)B^1(1)A^0(1)$ + $B_1^1A^0(1)$	$A^1(2)B_1^0B^0(2)$ + $A^1(2)A^1(1)B_1^0B^0(1)$	$A^1(2)A^1(1)B_1^0B^0(1)$ + $B_1^0B^0(2)$	$B_1^1A^0(2)+B_1^1A^0(1)+A^0(2)B^1(2)$ + $B^1(1)B^1(2)A^0(1)$
	5R		$A^1(2)B_1^0B^0(2)$ + $A^1(2)A^1(1)B_1^0B^0(1)$		$B_1^1A^0(2)+A^0(2)B^1(2)+B_1^1A^0(1)$ + $B^1(1)B^1(2)A^0(1)$
	6	$A^0(2)+B_1^1A^0(1)$ + $B^1(1)B^1(2)A^0(1)$	$A^1(2)B^0(2)B_1^0$ + $A^1(2)A^1(1)B^0(1)B_1^0$	$A^1(2)A^1(1)B^0(1)B_1^0$ + $B^0(2)B_1^0$	$B_1^1(A)^0(2)+B_1^1A^0(1)+A^0(2)B^1(2)$ + $B^1(2)B^1(1)A^0(1)$

Therefore:

$$C^1_{\beta_1} = A^0(1)B^0(1)$$

At t = 2:

$$C^1(2) = B^0(2)\beta_1^0 + A^1(2)A^1(1)B^0(1)\beta_1^0$$

$$C^0(2) = B^1(2)A^0(2) + B^1(2)B^1(1)A^0(1) + A^0(2)\beta_1^1 + A^0(1)\beta_1^1$$

Therefore:

$$C^2_{\beta_1} = A^0(2)B^0(2) + A^0(1)B^0(2)$$

TESTS FOR UNSPECIFIED FAULTS

Test sequences can be produced for individually inserted faults by generating the equations for the circuit and then finding the corresponding test functions using Equation (5). A more efficient strategy will now be described. The approach taken in the efficient strategy is to find test sequences for each primary input fault observed at each circuit output. Chappell [6] states that this strategy yields tests for 80-90% of the single faults in the circuit under consideration. The remaining faults can be handled using the previously described approach.

Generation of test sequences for faults on input I_i to be observed at output O_j is performed as follows. Let the following equation pair represent output O_j in terms of input I_i and other unspecified terms.

$$O_j^1 = A + BI_j^1 + CI_i^0$$

$$O_j^0 = D + EI_j^1 + FI_i^0$$

where A, B, C, D, E, F are Boolean expressions.

The desired test function is given below.

$$O_{I_i} = (BF + CE)\ (I_i^1 + I_i^0) \tag{6}$$

It should be emphasized that fault-free circuit equations are used in obtaining Equation (6).

The following test functions result for the cross-coupled NAND circuit for faults on input A with observation at output C.

t = 1:

$$C_A^1 = B^0(1)\ [A^1(1) + A^0(1)]$$

$$= A^1(1)B^0(1) + A^0(1)B^0(1)$$

t = 2:

$$C_A^2 = [B^0(2) + A^1(1)B^0(1)]\,[A^1(2) + A^0(2)]$$

$$= A^1(2)B^0(2) + A^0(2)B^0(2) + A^1(2)\ A^1(1)B^0(1)$$

$$+ A^0(2)A^1(1)B^0(1)$$

Selection of tests from (6) should be such that the input is exercised with both a logical 1 and a logical 0 if possible. Hence, two tests for each input-output pair are attempted. The shortest test in terms of time should be chosen when more than one possibility exists.

CHAPTER 4 - STARTING STATE SPECIFICATION

A major advantage of the logic model introduced in Chapter 2 is that a representation exists for describing a logic signal in an unknown state. This allows analysis of sequential circuits from an unknown starting state. However, sequential circuit examples have been found that cause difficulties when applying the analysis procedure of Chapter 2 if the starting states are unknown. This chapter will be devoted to a presentation of some of these difficulties. A possible approach to the solution of the problem is also given.

PROBLEM CIRCUITS

The circuit shown in Figure 4.1 is from [6] and has only one stable starting state. Establishment of this state must be accomplished manually before the analysis procedure of Chapter 2 can be applied. Hence, the unknown starting state will not yield meaningful results.

A possibly more important example of the inadequacies of the unknown starting state approach for sequential circuits is illustrated by the JK flip-flop circuit in Figure 4.2. Analysis of the JK flip-flop does not produce meaningful results when an unknown starting state is assumed due to the global feedback in the circuit. Table 4.1 shows the result of the analysis.

When a specific starting state is assumed for the JK flip-flop, the analysis procedure yields the proper results. Tables 4.2 and 4.3

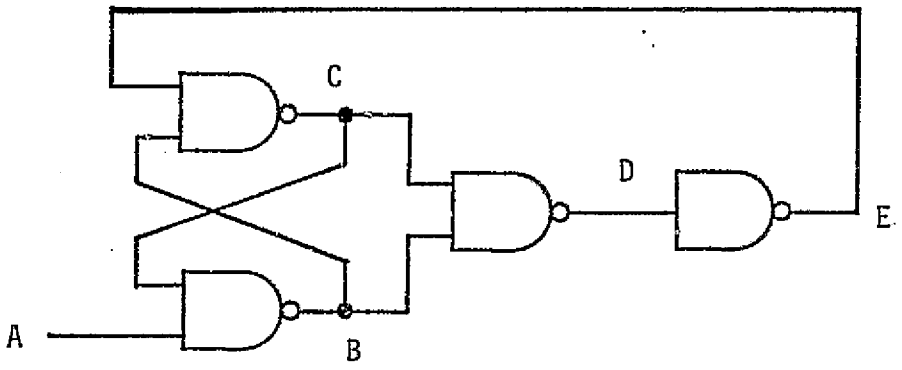


Figure 4.1. Self-Initializing Circuit [ ].

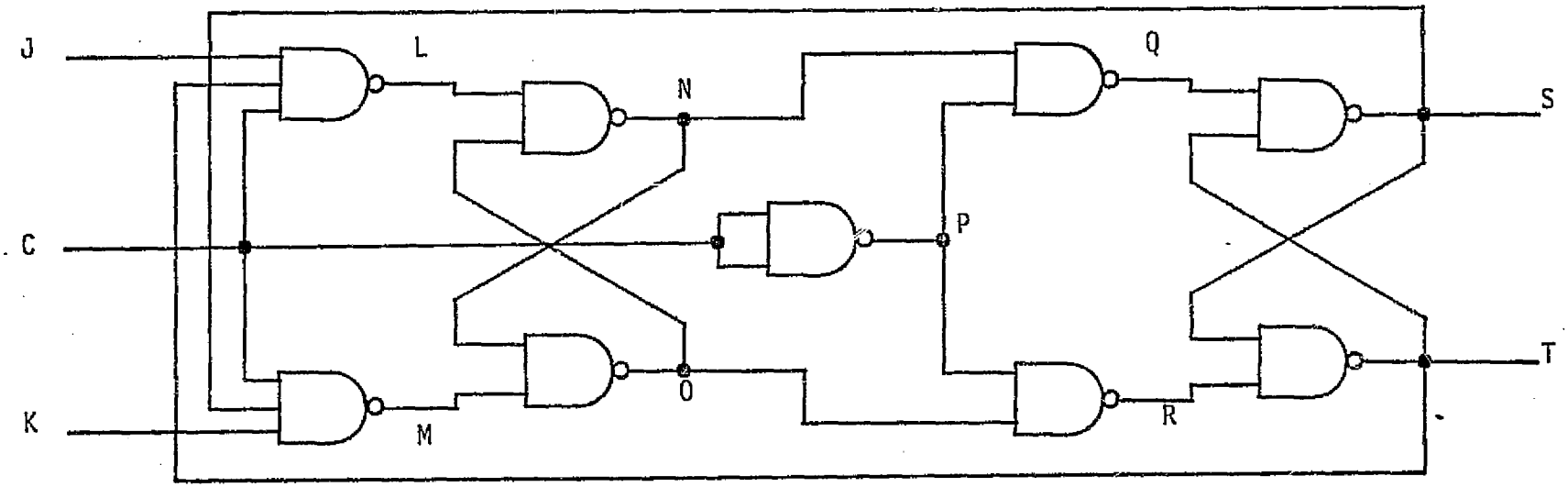


Figure 4.2. JK Flip-Flop Logic Circuit

Table 4.1

Equation Development for JK Flip-Flop with Unknown Starting State

$t = 0,\ r = 0$

$$L^1 = L^0 = M^1 = M^0 = N^1 = N^0 = O^1 = O^0 = P^1 = P^0 = Q^1 = Q^0 = R^1 = R^0 = S^1 = S^0 = T^1 = T^0 = 0$$

$t = 1,\ r = 1$

$L_1^1 = J_1^0 + C_1^0,$	$L_1^0 = 0$
$M_1^1 = K_1^0 + C_1^0,$	$M_1^0 = 0$
$N_1^1 = 0,$	$N_1^0 = 0$
$O_1^1 = 0,$	$O_1^0 = 0$
$P_1^1 = C_1^0,$	$P_1^0 = C_1^1$
$Q_1^1 = 0,$	$Q_1^0 = 0$
$R_1^1 = 0,$	$R_1^0 = 0$
$S_1^1 = 0,$	$S_1^0 = 0$
$T_1^1 = 0,$	$T_1^0 = 0$

$t = 1, r = 2$

$L_1^1 = J_1^0 + C_1^0$ $\quad L_1^0 = 0$

$M_1^1 = K_1^0 + C_1^0$ $\quad M_1^0 = 0$

$N_1^1 = 0$ $\quad N_1^0 = 0$

$O_1^1 = 0$ $\quad O_1^0 = 0$

$P_1^1 = C_1^0$ $\quad P_1^0 = C_1^1$

$Q_1^1 = C_1^1$ $\quad Q_1^0 = 0$

$R_1^1 = C_1^1$ $\quad R_1^0 = 0$

$S_1^1 = 0$ $\quad S_1^0 = 0$

$T_1^1 = 0$ $\quad T_1^0 = 0$

$t = 1, r = 3$

$L_1^1 = J_1^0 + C_1^0$ $\quad L_1^0 = 0$

$M_1^1 = K_1^0 + C_1^0$ $\quad M_1^0 = 0$

$N_1^1 = 0$ $\quad N_1^0 = 0$

$O_1^1 = 0$ $\quad O_1^0 = 0$

$P_1^1 = C_1^0$ $\quad P_1^0 = C_1^1$

$Q_1^1 = C_1^1$ $\quad Q_1^0 = 0$

$R_1^1 = C_1^1$ $\quad R_1^0 = 0$

$S_1^1 = 0$ $\quad S_1^0 = 0$

$T_1^1 = 0$ $\quad T_1^0 = 0$

$t = 2,\ r = 4$

$$L_2^1 = J_2^0 + C_2^0 \qquad L_2^0 = 0$$

$$M_2^1 = K_2^0 + C_2^0 \qquad M_2^0 = 0$$

$$N_2^1 = 0 \qquad N_2^0 = 0$$

$$O_2^1 = 0 \qquad O_2^0 = 0$$

$$P_2^1 = C_2^0 \qquad P_2^0 = C_2^1$$

$$Q_2^1 = C_1^1 \qquad Q_2^0 = 0$$

$$R_2^1 = C_1^1 \qquad R_2^0 = 0$$

$$S_2^1 = 0 \qquad S_2^0 = 0$$

$$T_2^1 = 0 \qquad T_2^0 = 0$$

$t = 2,\ r = 5$

$$L_2^1 = J_2^0 + C_2^0 \qquad L_2^0 = 0$$

$$M_2^1 = K_2^0 + C_2^0 \qquad M_2^0 = 0$$

$$N_2^1 = 0 \qquad N_2^0 = 0$$

$$O_2^1 = 0 \qquad O_2^0 = 0$$

$$P_2^1 = C_2^0 \qquad P_2^0 = C_2^1$$

$$Q_2^1 = C_2^1 \qquad Q_2^0 = 0$$

$$R_2^1 = C_2^1 \qquad R_2^0 = 0$$

$$S_2^1 = 0 \qquad S_2^0 = 0$$

$$T_2^1 = 0 \qquad T_2^0 = 0$$

$t = 2,\ r = 6$

$$L_2^1 = J_2^0 + C_2^0 \qquad L_2^0 = J_2^1 C_2^1$$

$$M_2^1 = K_2^0 + C_2^0 \qquad M_2^0 = K_2^1 C_2^1$$

$$N_2^1 = 0 \qquad N_2^0 = 0$$

$$O_2^1 = 0 \qquad O_2^0 = 0$$

$$P_2^1 = C_2^0 \qquad P_2^0 = C_2^1$$

$$Q_2^1 = C_2^1 \qquad Q_2^0 = 0$$

$$R_2^1 = C_2^1 \qquad R_2^0 = 0$$

$$S_2^1 = 0 \qquad S_2^0 = 0$$

$$T_2^1 = 0 \qquad T_2^0 = 0$$

TABLE 4.2

Equation Summary for JK Flip-Flop with 0 Starting State

$t = 0,\ r = 0$

$L_0^1 = 0$	$L_0^0 = 0$
$M_0^1 = 1$	$M_0^0 = 0$
$N_0^1 = 0$	$N_0^0 = 1$
$O_0^1 = 1$	$O_0^0 = 0$
$P_0^1 = 0$	$P_0^0 = 0$
$Q_0^1 = 1$	$Q_0^0 = 0$
$R_0^1 = 0$	$R_0^0 = 0$
$S_0^1 = 0$	$S_0^0 = 1$
$T_0^1 = 1$	$T_0^0 = 0$

$t = 1,\ r = 5$

$L_1^1 = J_1^0 + C_1^0$	$L_1^0 = J_1^1 C_1^1$
$M_1^1 = K_1^0 + C_1^0 + C_1^1$	$M_1^0 = 0$
$N_1^1 = J_1^1 C_1^1$	$N_1^0 = J_1^0 + C_1^0$
$O_1^1 + J_1^0 + C_1^0$	$O_1^0 = J_1^1 C_1^1$
$P_1^1 = C_1^0$	$P_1^0 = C_1^1$
$Q_1^1 = J_1^0 + C_1^0 + C_1^1$	$Q_1^0 = 0$
$R_1^1 = 0$	$R_1^0 = C_1^0$
$S_1^1 = 0$	$S_1^0 = C_1^0 + C_1^1$
$T_1^1 = C_1^0 + C_1^1$	$T_1^0 = 0$

t = 2, r = 12

$$L_2^1 = J_2^0 + C_2^0$$

$$L_2^0 = J_2^1C_2^1C_1^0 + J_2^1C_2^1C_1^1$$

$$M_2^1 = K_2^0 + C_2^0 + J_2^0J_1^0C_1^1 + J_2^0C_1^0 + C_2^1C_1^0 + C_2^1C_1^1$$

$$M_2^0 = 0$$

$$N_2^1 = K_2^0J_1^1C_1^1 + C_2^0J_1^1C_1^1 + J_2^1C_2^1C_1^0 + J_2^1C_2^1C_1^1 + C_2^1J_1^1C_1^1$$

$$N_2^0 + J_2^0J_1^0 + J_2^0C_1^0 + C_2^0J_1^0 + C_2^0C_1^0$$

$$O_2^1 = N_2^0$$

$$O_2^0 = N_2^1$$

$$P_2^1 = C_2^0$$

$$P_2^0 = C_2^1$$

$$Q_2^1 = J^0J^0 + J^0C^0 + C^0J^0 + C_2^0C_1^0 + C_2^1$$

$$Q_2^0 = C_2^0J_1^1C_1^1$$

$$R_2^1 = K_2^0J_1^1C_1^1 + C_2^0J_1^1C_1^1 + C_2^1$$

$$R_2^0 = C_2^0J_1^0 + C_2^0C_1^0$$

$$S_2^1 = C_2^0J_1^1C_1^1$$

$$S_2^0 = J_2^0J_1^0C_1^1 + J_2^0C_1^0 + C_2^0J_1^0 + C_2^0C_1^0 + C_2^1C_1^0 + C_2^1C_1^1$$

$$T_2^1 = S_2^0$$

$$T_2^0 = S_2^1$$

TABLE 4.3

Equation Summary for JK Flip-Flop with 1 Starting State

$t = 0,\ r = 0$

$L_0^1 = 1$ $\qquad L_0^0 = 0$

$M_0^1 = 0$ $\qquad M_0^0 = 0$

$N_0^1 = 1$ $\qquad N_0^0 = 0$

$O_0^1 = 0$ $\qquad O_0^0 = 1$

$P_0^1 = 0$ $\qquad P_0^0 = 0$

$Q_0^1 = 0$ $\qquad Q_0^0 = 0$

$R_0^1 = 1$ $\qquad R_0^0 = 0$

$S_0^1 = 1$ $\qquad S_0^0 = 0$

$T_0^1 = 0$ $\qquad T_0^0 = 1$

$t = 1,\ r = 5$

$L_1^1 = J_1^0 + C_1^0 + C_1^1$ $\qquad L_1^0 = 0$

$M_1^1 = K_1^0 + C_1^0$ $\qquad M_1^0 = K_1^1 C_1^1$

$N_1^1 = K_1^0 + C_1^0$ $\qquad N_1^0 = K_1^1 C_1^1$

$O_1^1 = K_1^1 C_1^1$ $\qquad O_1^0 = K_1^0 + C_1^0$

$P_1^1 = C_1^0$ $\qquad P_1^0 = C_1^1$

$Q_1^1 = C_1^1$ $\qquad Q_1^0 = C_1^0$

$R_1^1 = K_1^0 + C_1^0 + C_1^1$ $\qquad R_1^0 = 0$

$S_1^1 = C_1^0 + C_1^1$ $\qquad S_1^0 = 0$

$T_1^1 = 0$ $\qquad T_1^0 = C_1^0 + C_1^1$

t = 2, r = 12

$$L_2^1 = J_2^0 + C_2^0 + K_2^0K_1^0C_1^1 + K_2^0C_1^0 + C_2^1C_1^0 + C_2^1C_1^1$$

$$L_2^0 = 0$$

$$M_2^1 = K_2^0 + C_2^0$$

$$M_2^0 = K_2^1C_2^1C_1^0 + K_2^1C_2^1C_1^1$$

$$N_2^1 = K_2^0K_1^0 + K_2^0C_1^0 + C_2^0K_1^0 + C_2^0C_1^0$$

$$N_2^0 = J_2^0K_1^1C_1^1 + C_2^0K_1^1C_1^1 + K_2^1C_2^1C_1^0 + K_2^1C_2^1C_1^1 + C_2^1K_1^1C_1^1$$

$$O_2^1 = N_2^0$$

$$O_2^0 = N_2^1$$

$$P_2^1 = C_2^0$$

$$P_2^0 = C_2^1$$

$$Q_2^1 = J_2^0K_1^1C_1^1 + C_2^0K_1^1C_1^1 + C_2^1$$

$$Q_2^0 = C_2^0K_1^0 + C_2^0C_1^0$$

$$R_2^1 = K_2^0K_1^0 + K_2^0C_1^0 + C_2^0K_1^0 + C_2^0C_1^0 + C_2^1$$

$$R_2^0 = C_2^0K_1^1C_1^1$$

$$S_2^1 = K_2^0K_1^0C_1^1 + K_2^0C_1^0 + C_2^0K_1^0 + C_2^0C_1^0 + C_2^1C_1^0 + C_2^1C_1^1$$

$$S_2^0 = C_2^0K_1^1C_1^1$$

$$T_2^1 = S_2^0$$

$$T_2^0 = S_2^1$$

summarize these results for a 0 starting state and a 1 starting state, respectively.

The analysis of large sequential circuits cannot be accomplished effectively unless the analysis procedure can start from an unknown starting condition. An approach will be described in the next section that holds promise for solving the starting state problem for the procedure given in Chapter 2.

PROPOSED PROBLEM SOLUTION

Consider the cross-coupled NAND gates shown in Figure 4.3. In a physical circuit, the state will either be C=0 and D=1 or C=1 and D=0 when stable. Hence, it is reasonable to assume that the starting outputs of any pair of corss-coupled NAND gates are complementary even though they may be unknown. Therefore, it appears reasonable to use a variable to represent the starting state of cross-coupled NAND gates. In terms of variable pairs, the starting conditions would be specified as follows.

$$C^1 = X$$
$$C^0 = \overline{X}$$
$$D^1 = \overline{X}$$
$$D^0 = X$$

Table 4.4 shows the analysis of the NAND circuit in terms of the above starting state. As can be seen, the results are compatible with those obtained using constants to specify the starting state.

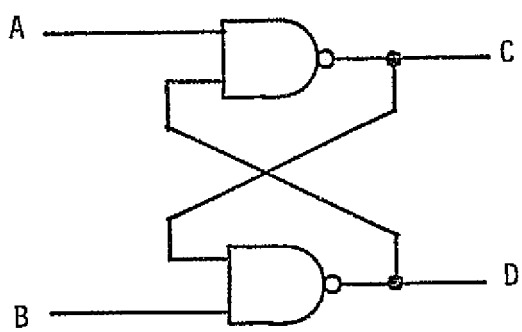


Figure 4.3. Cross-Coupled NAND Gate Pair

TABLE 4.4

Equation Development Using Starting State Variables

$t = 0,\ r = 0$

$C_0^1 = X$ $\qquad C_0^0 = \overline{X}$

$D_0^1 = \overline{X}$ $\qquad D_0^0 = X$

$t = 1,\ r = 1$

$C_1^1 = A_1^0 + X$ $\qquad C_1^0 = A_1^1\overline{X}$

$D_1^1 = B_1^0 + \overline{X}$ $\qquad D_1^0 = B_1^1 X$

$t = 1,\ r = 2$

$C_1^1 = A_1^0 + B_1^1 X$ $\qquad C_1^0 = A_1^1 B_1^0 + A_1^1\overline{X}$

$D_1^1 = B_1^0 + A_1^1\overline{X}$ $\qquad D_1^0 = A_1^0 B_1^1 + B_1^1 X$

$t = 1,\ r = 3$

$C_1^1 = A_1^0 + B_1^1 X$ $\qquad C_1^0 = A_1^1 B_1^0 + A_1^1\overline{X}$

$D_1^1 = B_1^0 + A_1^1\overline{X}$ $\qquad D_1^0 = A_1^0 B_1^1 + B_1^1 X$

$t = 2,\ r = 4$

$C_2^1 = A_2^0 + A_1^0 B_1^1 + B_1^1 X,$ $\qquad C_2^0 = A_2^1 B_2^0 B_1^0 + A_2^1 A_1^1 B_1^0 + A_2^1 A_1^1\overline{X}$

$D_2^1 = B_2^0 + A_1^1 B_1^0 + A_1^1\overline{X},$ $\qquad D_2^0 = A_2^0 B_2^1 A_1^0 + B_2^1 A_1^0 B_1^1 + B_2^1 B_1^1 X$

$\underline{t = 2,\ r = 5}$

$$C_2^1 = A_2^0 + B_2^1 A_1^0 B_1^1 + B_2^1 B_1^1 X, \qquad C_2^0 = A_2^1 B_2^0 + A_2^1 A_1^1 B_1^0 + A_2^1 A_1^1 \overline{X}$$

$$D_2^1 = B_2^0 + A_2^1 A_1^1 B_1^0 + A_2^1 A_1^1 \overline{X}, \qquad D_2^0 = A_2^0 B_2^1 + B_2^1 A_1^0 B_1^1 + B_2^1 B_1^1 X$$

$\underline{t = 2,\ r = 6}$

$$C_2^1 = A_2^0 + B_2^1 A_1^0 B_1^1 + B_2^1 B_1^1 X, \qquad C_2^0 = A_2^1 B_2^0 + A_2^1 A_1^1 B_1^0 + A_2^1 A_1^1 \overline{X}$$

$$D_2^1 = B_2^0 + A_2^1 A_1^1 B_1^0 + A_2^1 A_1^1 \overline{X}, \qquad D_2^0 = A_2^0 B_2^1 + B_2^1 A_1^0 B_1^1 + B_2^1 B_1^1 X$$

The use of variables to specify the starting state of cross-coupled NAND gates appears to offer promise as a solution to the starting state problem for sequential circuits. However, when using the approach on the JK flip-flop circuit of Figure 4.2 a problem was encountered when applying the race analysis procedure described in Chapter 2. More specifically, as now organized the race analysis procedure eliminates terms during equation modification that should be retained. Therefore, the race analysis procedure must be revised to accommodate the new approach.

CHAPTER 5 - CONCLUSIONS AND RECOMMENDATIONS

An approach to test pattern generation has been described that can be used with both combinational and sequential logic circuits. Two strategies can be used to obtain test patterns for single stuck-at faults. One strategy is used to find test patterns for a specific fault that has been inserted in the circuit. The other strategy is used to obtain a set of tests for unspecified faults in an attempt to reduce the computation time per fault. Combined, the two strategies provide an effective approach to test pattern generation for large logic circuits.

The procedures presented in this report have been adopted from the concepts outlined by Chappell [6]. A major modification has been made in the race analysis procedure, however.

The major deficiency found in the approach presented is the problem of starting state specification for some sequential circuits such as the JK flip-flop. A possible solution to this problem has been suggested but needs further work.

It is recommended that future work on the test pattern generation procedure first be directed toward finding a solution to the starting state problem. A production program of the procedure should be developed after this problem has been solved. APL or FORTRAN should be considered as the language for use in the program.

Other problem areas needing attention are generalization of the timing model and functional representation of circuit segments. The latter is one approach that could possibly reduce computation time and memory requirements of the procedure.

Further work on oscillation analysis is also in order. A means for precisely setting an upper limit on the number of ripple time steps per input time step is needed. The problem of immediate detection of an oscillation condition is also worth of attention. Development of a rule for halting oscillation is needed.

REFERENCES

1. W. G. Bouricius,et.al.; " Algorithms for Detection of Faults in Logic Circuits," IEEE-TC, Vol. C-20,No. 11, November 1971,pp.1258-1264.

2. M. Y. Hsiao and D. K. Chia, "Boolean Difference for Fault Detection in Asynchronous Sequential Machines," IEEE-TC, Vol. C-20, No. 11, November 1971, pp. 1356-1361.

3. M. A. Breuer, "A Random and an Algorithmic Technique for Fault Detection Test Generation for Sequential Circuits," IEEE-TC, Vol.C-20, No. 11, November 1971, pp. 1364-1370.

4. V. D. Agrawal and P. Agrawal, " An Automatic Test Generation System for Illiac IV Logic Board," IEEE-TC, Vol. C-21, No. 9, September 1972, pp. 1015-1017.

5. M. A. Breuer, "Testing for Intermittent Faults in Digital Circuits," IEEE-TC, Vol. C-22, No. 3, March 1973, pp. 241-246.

6. S.G. Chappell, "Automatic Test Generation for Asynchronous Digital Circuits," Bell System Technical Journal, Vol. 53, No. 8, October 1974, pp. 1477-1503.

APPENDIX

Description of SIMLOG

FUNCTIONAL DESCRIPTION

This appendix is devoted to the description of a computer program called SIMLOG that performs the analysis procedure described in Chapter 2. A functional level flowchart of the program is given in Figure A.1. The program has been written in Xerox BASIC for running under the CP-V Operating System on an XDS Sigma 5 Computer. Evaluation of test sequences produced by the TPG procedure and analysis of planned modifications of the procedure have provided the impetus for writing the program. Production runs of the program are not anticipated. However development of a production version of the program is expected after completion of the current evaluation and analysis studies.

Program Capabilities

Circuits consisting of NAND and NOT gates can be accommodated by SIMLOG. The sum of the number of circuit inputs plus the number of gates must not exceed twenty-six. Starting states of each logic signal may be specified as 0 or 1 or may be left unspecified. Propagation of starting conditions through a circuit must be accomplished manually.

The program generates the time dependent equation pairs for the outputs of all gates in a circuit. Race analysis is performed on each set of cross-coupled NAND gates as discussed previously. The race analysis is performed interactively with the program performing all computation and the user making decisions about equation changes upon cue from the program. Ripple time is incremented automatically until a stable set of equations is reached. Input time incrementing is under user control but must not exceed a value of nine.

Fault insertion has not been incorporated in the program at this time. However, a fault insertion capability will be added in the immediate future. Oscillation analysis other than for races is also currently lacking in the program. The addition of oscillation analysis is also planned. Computation of fault functions is not included in this program. Addition of this step is not anticipated since it can be easily carried out manually for circuits of the size handled by the program.

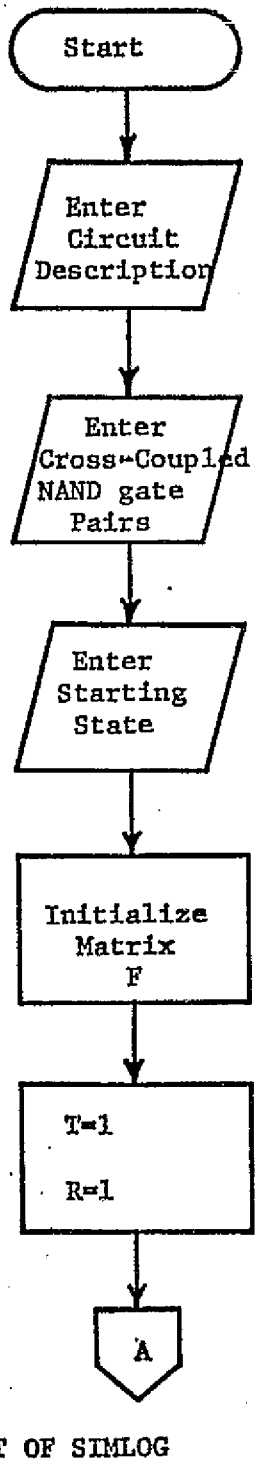


Fig. A.1. FUNCTIONAL FLOWCHART OF SIMLOG

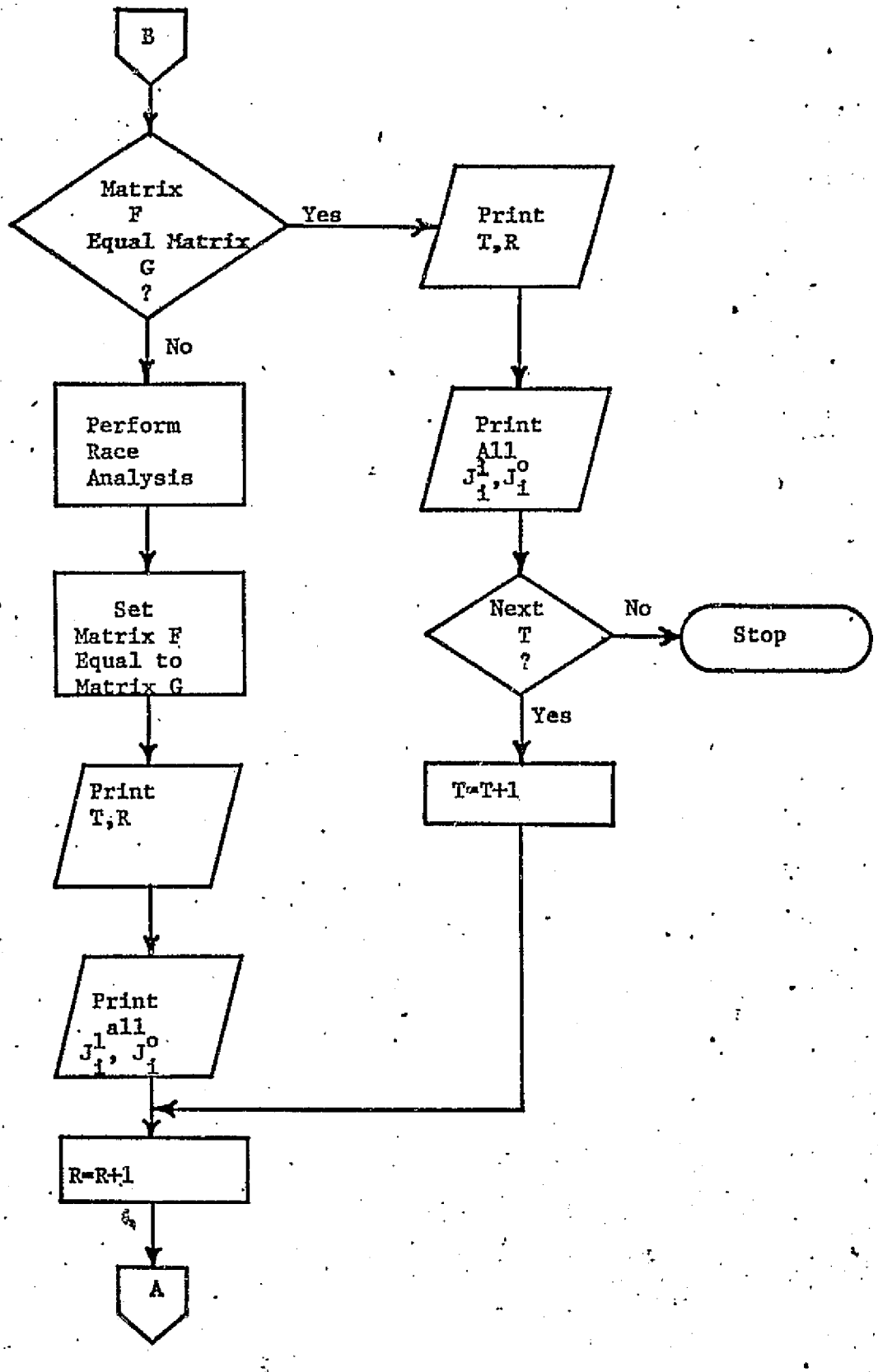


Fig. A.1 (Continued)

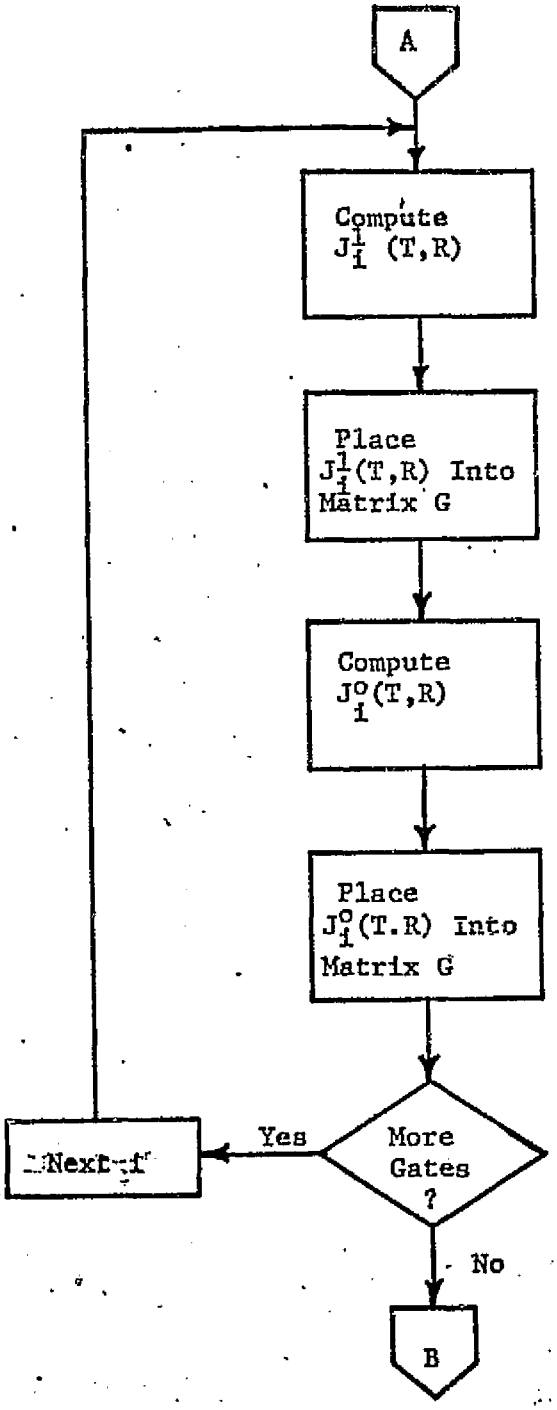


Fig. A.1 (Continued)

User Procedure

The steps listed below outline the procedure followed by a user of SIMLOG for a typical logic circuit.

1. Draw a logic diagram and label each circuit input and each gate output with a unique letter of the alphabet.

2. Call SIMLOG and enter RUN.

3. Enter the circuit description when requested by typing the letter associated with each gate output followed by the letters of the corresponding gate inputs. Letters corresponding to circuit inputs must be followed by a minus sign. Enter a 0 when finished.

4. Enter each cross-coupled NAND gate pair. Terminate entry by typing 0.

5. Enter starting states of each logic signal as desired. The default condition is the unknown state. Again terminate by typing 0.

6. Respond to the race analysis procedure by typing YES or NO as appropriate.

7. Select the next input time step by typing YES when requested. A NO response terminates program execution.

8. Obtain a hard copy output of all equation sets by copying the contents of file PRINTOUT to the line printer using the file manipulation features of the CP-V monitor or PCL.

A sample computer run of the program for a pair of cross-coupled NAND gates is shown in Figure A.2. The equations produced by the program for a JK flip-flop assumed to be in starting state 0 are given in Figure A.3.

Program Expansion

The following features will be added to the current program in the near future.

1. Fault insertion

2. Variables for representing initial conditions

3. Totally automatic race analysis

4. Oscillation analysis

```
09:19   AUG 07  SIMLOG...
ENTER GATE OUTPUT SYMBOL FOLLOWED BY GATE INPUT SYMBOLS.
NETWORK INPUT SYMBOLS SHOULD BE FOLLOWED BY A MINUS SIGN.
WHEN ALL GATES HAVE BEEN ENTERED--ENTER 0.
?CA-D
?DB-C
?0
ENTER CROSS-COUPLED GATE PAIRS--ENTER 0 TO STOP.
?CD
?0
ENTER STARTING STATE(0 OR 1) FOR EACH SIGNAL LINE BY
TYPING THE LINE SYMBOL FOLLOWED BY =0 OR =1.
TO TERMINATE ENTER 0.
SIGNAL LINES NOT ENTERED ARE TREATED AS UNKNOWN CONDITIONS.
?0

T=   1       R=   1

C11=A01

C01=0

D11=B01

D01=0


BEGIN RACE ANALYSIS.

C0.= 0

C0.=A11B01

FUNCTIONS IDENTICAL   ?NO
D1.=B01

D1.=B01

FUNCTIONS IDENTICAL   ?YES
D0.= 0

D0.=A01B11

FUNCTIONS IDENTICAL   ?NO
C1.=A01

C1.=B01

FUNCTIONS IDENTICAL   ?YES
```

Fig. A.2 SIMLOG Output for Cross-Coupled NAND Gates

```
T=  1        R=  2
C11=A01
C01=A11B01
D11=B01
D01=A01B11

T=  1        R=  3
C11=A01
C01=A11B01
D11=B01
D01=A01B11
NEXT T  ?YES
BEGIN RACE ANALYSIS.
C0.=A11B01
C0.=A12B01
FUNCTIONS IDENTICAL   ?NO
D1.=B01
D1.=B02+A11B01
FUNCTIONS IDENTICAL   ?NO
T=  2        R=  4
C12=A02+A01B11
C02=A12B01B02+A12A11B01
D12=B02+A11B01
D02=A01A02B12+A01B12B11
```

Fig. A.2 (Continued)

```
BEGIN RACE ANALYSIS.

C0.=A12B01B02+A12A11B01

C0.=A12B02+A12A11B01

FUNCTIONS IDENTICAL   ?NO
D1.=B02+A11B01

D1.=B02+A12A11B01

FUNCTIONS IDENTICAL   ?NO

T=  2        R=  5

C12=A02+A01B12B11

C02=A12B02+A12A11B01

D12=B02+A12A11B01

D02=A02B12+A01B12B11


T=  2        R=  6

C12=A02+A01B12B11

C02=A12B02+A12A11B01

D12=B02+A12A11B01

D02=A02B12+A01B12B11

NEXT T  ?NO
```

Fig. A.2 (Continued

```
M11=1                          T=   1          K=   1
                               L11=J01+C01
M01=0
                               L01=C11J11
N11=0
N01=0
θ11=1
θ01=0
P11=C01
P01=C11
Q11=1
Q01=0
R11=0
R01=0
S11=0
S01=1
T11=1
T01=0
T=   1          K=   2
L11=J01+C01
L01=C11J11
M11=1
M01=0
N11=C11J11
N01=J01+C01
θ11=J01+C01 ←
θ01=0
P11=C01
P01=C11
Q11=C11
```


Fig. A.3

Q01=0

R11=C11

R01=C01

S11=0

S01=1

T11=1

T01=0

T= 1 R= 3

L11=J01+C01

L01=C11J11

M11=1

M01=0

N11=C11J11

N01=J01+C01

B11=J01+C01

B01=C11J11

P11=C01

P01=C11

Q11=J01+C01+C11

Q01=0

R11=C11

R01=C01

S11=0

S01=C11

T11=1

T01=0

T= 1 R= 4

L11=J01+C01

L01=C11J11

```
M11=K01+C01+C11
M01=0
N11=C11J11
N01=J01+C01
B11=J01+C01
B01=C11J11
P11=C01
P01=C11
Q11=J01+C01+C11
Q01=0
R11=C11
R01=C01
S11=0
S01=C01+C11
T11=C01+C11
T01=0
T=    1          R=    5
L11=J01+C01
L01=C11J11
M11=K01+C01+C11
M01=0
N11=C11J11
N01=J01+C01
B11=J01+C01
B01=C11J11
P11=C01
P01=C11
Q11=J01+C01+C11
```

```
Q01=0
R11=C11
R01=C01
S11=0
S01=C01+C11
T11=C01+C11
T01=0
T=    2          R=    6
L12=J02+C02
L02=C12C01J12+C12C11J12
M12=K02+CJ2+C01+C11
M02=0
N12=C11J11
N02=J01+C01
B12=J01+C01
B02=C11J11
P12=C02
P02=C12
Q12=J01+C01+C11
Q02=0
R12=C11
R02=C01
S12=0
S02=C01+C11
T12=C01+C11
T02=0
T=    2          R=    7
L12=J02+C02
```

```
L02=C12C01J12+C12C11J12
M12=K02+C02+C01+C11
M02=0
N12=C12C01J12+C12C11J12+C11J11
N02=J02J01+C01J02+C02J01+C02C01
012=J01+C01
002=C11J11
P12=C02
P02=C12
Q12=J01+C01+C12
Q02=C11C02J11
R12=C11J11+C12
R02=C02J01+C01C02
S12=0
S02=C01+C11
T12=C01+C11
T02=0
T=    2          R=    3
L12=J02+C02
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
L02=C12C01J12+C12C11J12
M12=K02+C02+C01+C11
M02=0
N12=C12C01J12+C12C11J12+C11J11
N02=J02J01+C01J02+C02J01+C02C01
012=J02J01+C01J02+C02J01+C02C01
002=C01C12J12+C11C12J12+C11J11
P12=C02
P02=C12
```

Q12=J02J01+C01J02+C02J01+C02C01+C12

QO2=C11C02J11

R12=C11J11+C12

RO2=C02J01+C01C02

S12=C11C02J11

SO2=C11J01+C01+C12C11

T12=C02J01+C01+C11

TO2=0

T= 2 R= 9

L12=J02+C02

LO2=C12C01J12+C12C11J12

M12=K02+C02+C11J01+C01+C12C11

MO2=0

N12=C01C12J12+C11C12J12+C11J11

NO2=J02J01+C01J02+C02J01+C02C01

O12=J02J01+C01J02+C02J01+C02C01

OO2=C01C12J12+C11C12J12+C11J11

P12=C02

PO2=C12

Q12=J02J01+C01J02+C02J01+C02C01+C12

QO2=C11C02J11

R12=C11J11+C12

RO2=C02J01+C02C01

S12=C11C02J11

SO2=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11

T12=C02J01+C11J01+C01+C12C11

TO2=C11C02J11

T= 2 R= 10

L12=J02+C02

```
L02=C12C01J12+C12C11J12
M12=K02+C02+C11J02J01+C01J02+C12C01+C12C11
M02=0
N12=C01C12J12+C11C12J12+C11J11
N02=J02J01+C01J02+C02J01+C02C01
B12=J02J01+C01J02+C02J01+C02C01
B02=C11J11K02+C02C11J11+C01C12J12+C12C11J12+C12C11J11
P12=C02
P02=C12
Q12=J02J01+C01J02+C02J01+C02C01+C12
Q02=C11C02J11
R12=C11J11+C12
R02=C02J01+C02C01
S12=C11C02J11
S02=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T12=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T02=C11C02J11
T=    2          R=    11
L12=J02+C02
L02=C12C01J12+C12C11J12
M12=K02+C02+C11J02J01+C01J02+C12C01+C12C11
M02=0
N12=C11J11K02+C02C11J11+C01C12J12+C12C11J12+C12C11J11
N02=J02J01+C01J02+C02J01+C02C01
B12=J02J01+C01J02+C02J01+C02C01
B02=C11J11K02+C02C11J11+C12C01J12+C12C11J12+C12C11J11
P12=C02
P02=C12
```

```
Q12=J02J01+C01J02+C02J01+C02C01+C12
Q02=C11C02J11
R12=C11J11K02+C02C11J11+C12
R02=C02J01+C02C01
S12=C11C02J11
S02=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T12=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T02=C11C02J11
T=    2          R=    12
L12=J02+C02
L02=C12C01J12+C12C11J12
M12=K02+C02+C11J02J01+C01J02+C12C01+C12C11
M02=0
N12=C11J11K02+C02C11J11+C12C01J12+C12C11J12+C12C11J11
N02=J02J01+C01J02+C02J01+C02C01
O12=J02J01+C01J02+C02J01+C02C01
O02=C11J11K02+C02C11J11+C12C01J12+C12C11J12+C12C11J11
P12=C02
P02=C12
Q12=J02J01+C01J02+C02J01+C02C01+C12
Q02=C02C11J11
R12=C11J11K02+C02C11J11+C12
R02=C02J01+C02C01
S12=C11C02J11
S02=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T12=C11J02J01+C01J02+C02J01+C02C01+C12C01+C12C11
T02=C02C11J11
T=    2          R=    13
```

DETAILED DESCRIPTION

SIMLOG is composed of a main program and nine subroutines. Each of these components is discussed in detail below. A top level flowchart of the program is given in Fig.A.4.

Main Program

The main program performs input, bookkeeping, subroutine calls and logical control functions as required by the logic-circuit analysis procedure. Fig.A.4 serves a flowchart of the main program. Array structures and variables used in the program will now be described.

N(27,27) - This array represents the interconnection pattern of the logic circuit. Columns 1-26 correspond to letters A-Z of the alphabet, respectively. Column 27 contains the decimal equivalent of the EBCDIC representation of the gate output symbol for the gate corresponding to the row. Each gate is assigned to a row of the array in the same order as the gate is entered during the circuit input operation. Gate inputs are indicated by a 1 or a-1 in the appropriate row and column. An input from another gate output is indicated by a1. A circuit input is indicated by a-1. End of data is indicated by an all zero r v.

M(5,2) - Cross-coupled NAND pairs are indicated in this array. Each pair corresponds to a column of the array. An all zero column is used to indicate the end of data.

F(100,27) - The time dependent equation set for ripple time r-1 is stored in this array. Location of each function is determined by pointers stored in array L to be discussed later.

Each product term of a function uses one or more rows in the array. The presence of a variable in a term is indicated by a non-zero entry in the corresponding column of the array. A positive entry represents a

Fig. A.4a - Toplevel Flowchart of SIMLOG

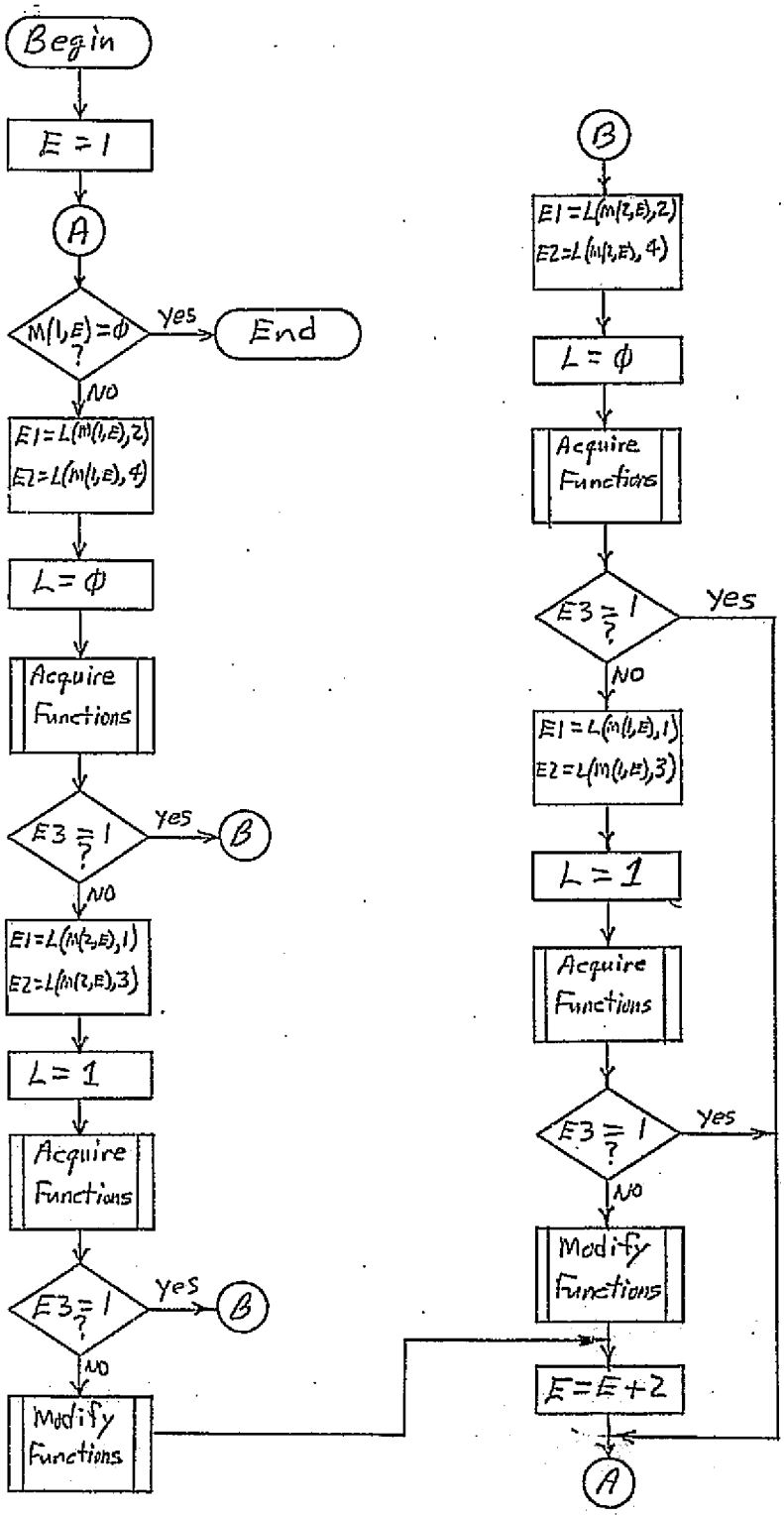


Fig. A.4b - Race Analysis Procedure

one variable, while a negative entry represents a zero variable. The input time associated with a variable is indicated by the absolute value of the entry. Column 27 is used to indicate the end of term or end of function. A 1 in column 27 indicates that the term is continued in the next row. A-1 in column 27 indicates that a new term starts in the next row. A number < = -90 indicates end of function. The absolute value of the end of function number indicates the function name in EBCDIC.

G(100,27) - This array stores the time dependent functions being computed at ripple time r. It has the same structure as F.

H(100,27) - This array has the same structure as F and is used as a scratch pad during function computation.

T(100,27) - Contents of G are temporarily stored in this array during the updating of functions in G.

L(26,4) - Pointers to the functions stored in arrays F and G are kept in this array. Columns 1 and 2 point to F, while columns 3 and 4 point to G. One function pointers are contained in columns 1 and 3. Zero function pointers are contained in columns 2 and 4. Pointers are stored in the row corresponding to the function name. Unused words contain ones.

A(10) - Used during input operations.

E - Index used to sequence through M array during race analysis.

E1 - Pointer to function in F matrix to be displayed during race analysis.

E2 - Pointer to function in G matrix to be displayed during race analysis.

H - Index used during loading of H matrix.

I - Flag used to indicate whether a one function (I=1) or a zero function (I=0) is being formed.

J - Logical complement of I.

K - Flag used to indicate first pass (K=1) through equation forming process for each equation.

L - Flag used to indicate one functions (L=1) or zero functions (L=0) are to be compared during race analysis.

P - Index used to sequence through columns of N array.

R - Ripple time variable.

S - Index used to sequence through rows of N array.

I - Input time variable.

Z9 - Flag set (Z9=1)when F and G arrays are identical.

OR Subroutine

The functions of this subroutine is to perform the logical OR of two Boolean functions. Functions to be OR'ed are placed in the H array by the main program. The OR subroutine performs the OR operation and passes control to the MIN subroutine if neither function is the identity, then control is returned to the Main program without going to MIN. Control is returned to OR from MIN. OR then returns control to Main program. The function resulting is placed in H.

All arrays used by OR have been previously described. The following variables have particular significance to OR.

X - Index for sequencing through rows of H array.

X1 - Variable indicating the number of rows in H array required to describe the minimized function. Value is fixed in MIN.

H - H array index. Value after OR is one greater than number of

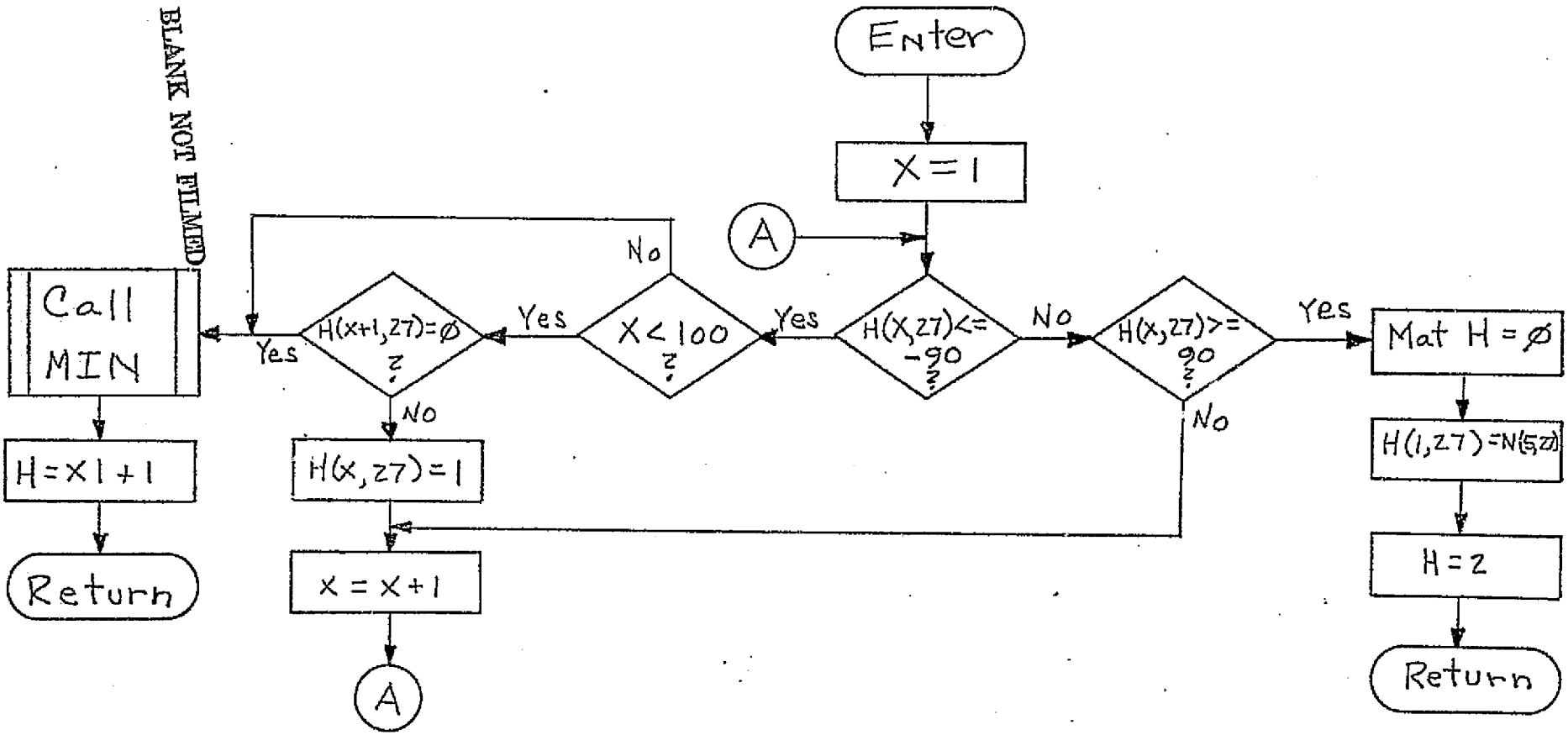


FIG. A.5 OR SUBROUTINE FLOWCHART

rows of H array needed to describe minimized function.

AND Subroutine

The AND subroutine performs the logical AND of two Boolean functions. The functions to be AND'ed are placed in the H array by the calling program. Results of the ANDing operation are placed in the top of the H array for return to the calling program. Fig.A.6 gives a functional flowchart of the AND subroutine.

Arrays used in AND have been previously defined. A list of variables used in the subroutine is given below.

A - Dummy index used in array element manipulation.

A1 - Dummy index used in array element manipulation.

C1 - Counter used during move of final product to top of H array.

X0 - Dummy index variable used during merging of terms and removal of duplicate variables.

X1 - Index for specifying terms in first functions.

X2 - Index for scanning rows of term indexed by X1.

X3 - Index for specifying terms of second function.

X4 - Index for scanning rows of term indexed by X3.

X5 - Index for specifying terms of the product function.

X6 - Index for specifying rows of terms specified by X5.

X7 - Index used during removal of redundant variables from terms.

X8 - Index used during removal of redundant variable.

Y - Index used for scanning columns.

Z3 - Variable indicating locations of the beginning of the first term in second function.

Z4 - Index used during moving of final product to top of H array.

Z5 - Variable indicating location of the beginning of the first term of the product function.

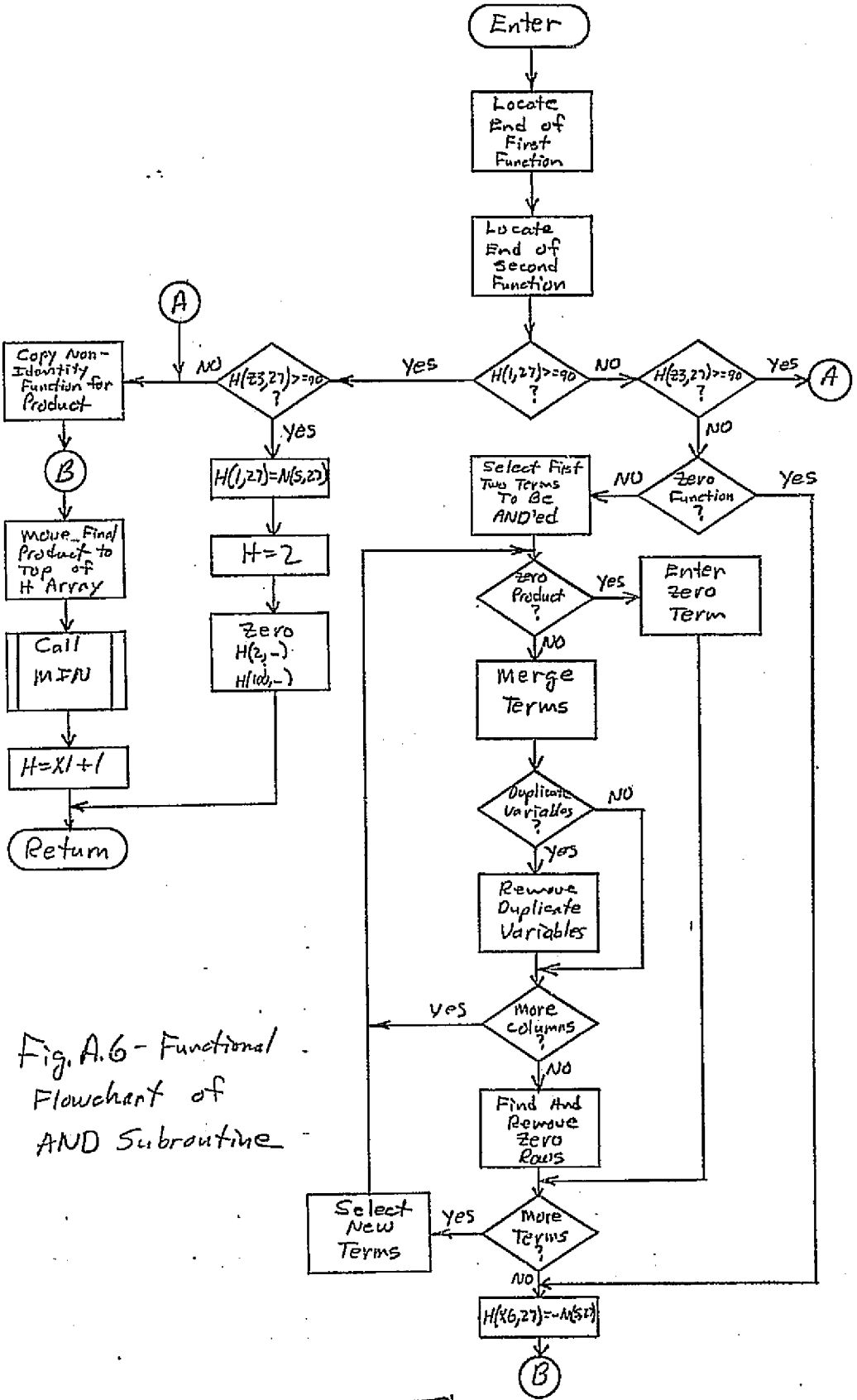


Fig. A.6 - Functional Flowchart of AND Subroutine

MIN Subroutine

This subroutine performs Boolean minimization of the types below.

$T + T = T$

$T + ST = T$

where S and T are Boolean product terms. A functional flowchart of the routine is given in Fig. A. 7.

The H array used in the routine is the same as described previously. Variables with special significance are defined below.

C - Counter used to indicate number of common variables shared by two terms for a given column.

C1 - Flag used to indicate when a match has been found among variables of two terms.

W1 - Flag indicating whether term one is zero (W1 = 0) or non-zero (W1 = 1).

W2 - Same as W1 for term two.

X1 - First row of first term.

X2 - First row of second term.

X3 - Index used to scan rows of first term.

X4 - Index used to scan rows of second term.

Y - Index used to scan columns of terms being compared.

Z - Mode of comparison of terms.

0 : Terms not comparable

1 : Term one is redundant

2 : Term two is redundant

3 : Terms are equal

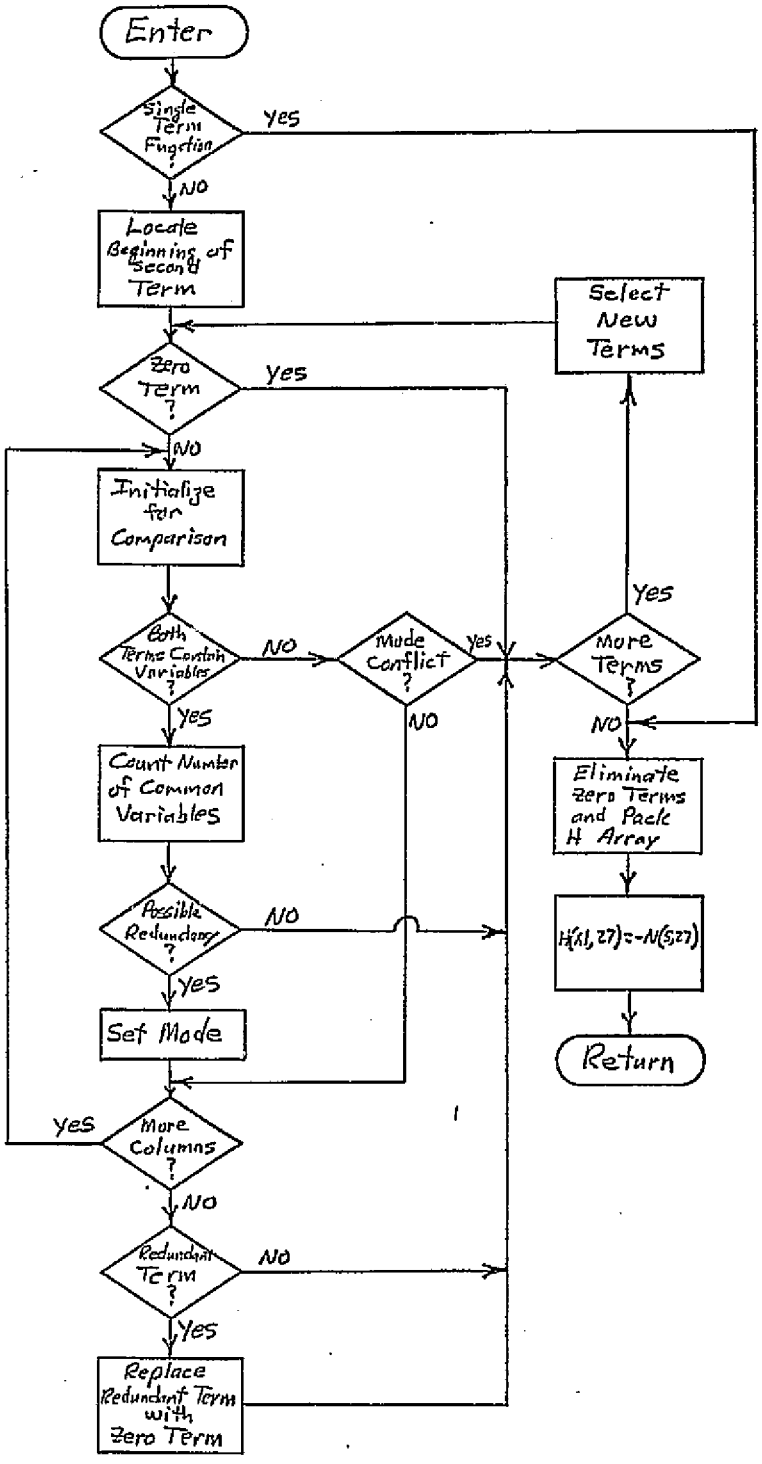


Fig. A.7 – Functional Flowchart of MIN Subroutine

OUTPUT Subroutine

The purpose of this subroutine is to format the functions stored in the F array into a character string and to print the input time, ripple time, and set of functions. Fig. A.8 shows the flowchart of the subroutine.

Arrays not previously encountered are described below.

P(500) - Functional information stored in F is translated into decimal character codes (EBCDIC) equivalents and stored in P.

Q(72) - Each individual function stored in F is transferred one-at-a-time to Q. The contents of Q are changed to character format and then is printed.

FUNCTION IDENTITY CHECK Subroutine

This subroutine is used during race analysis and performs the function of acquiring a time r-1 function from F and the corresponding time r function from G. The subroutine calls another subroutine to OUTPUT the functions. The subroutine then queries the user about equality of the two functions and returns control to the main program.

FUNCTION IDENTITY CHECK-OUTPUT Subroutine

Performs output of functions to be checked for equality during race analysis. The subroutine is a modification of the PRINTOUT subroutine.

FUNCTION MODIFICATION Subroutine

The purpose of this routine is to perform equation modification as specified in the race analysis procedure when a possible race condition has been detected. Called by Main program.

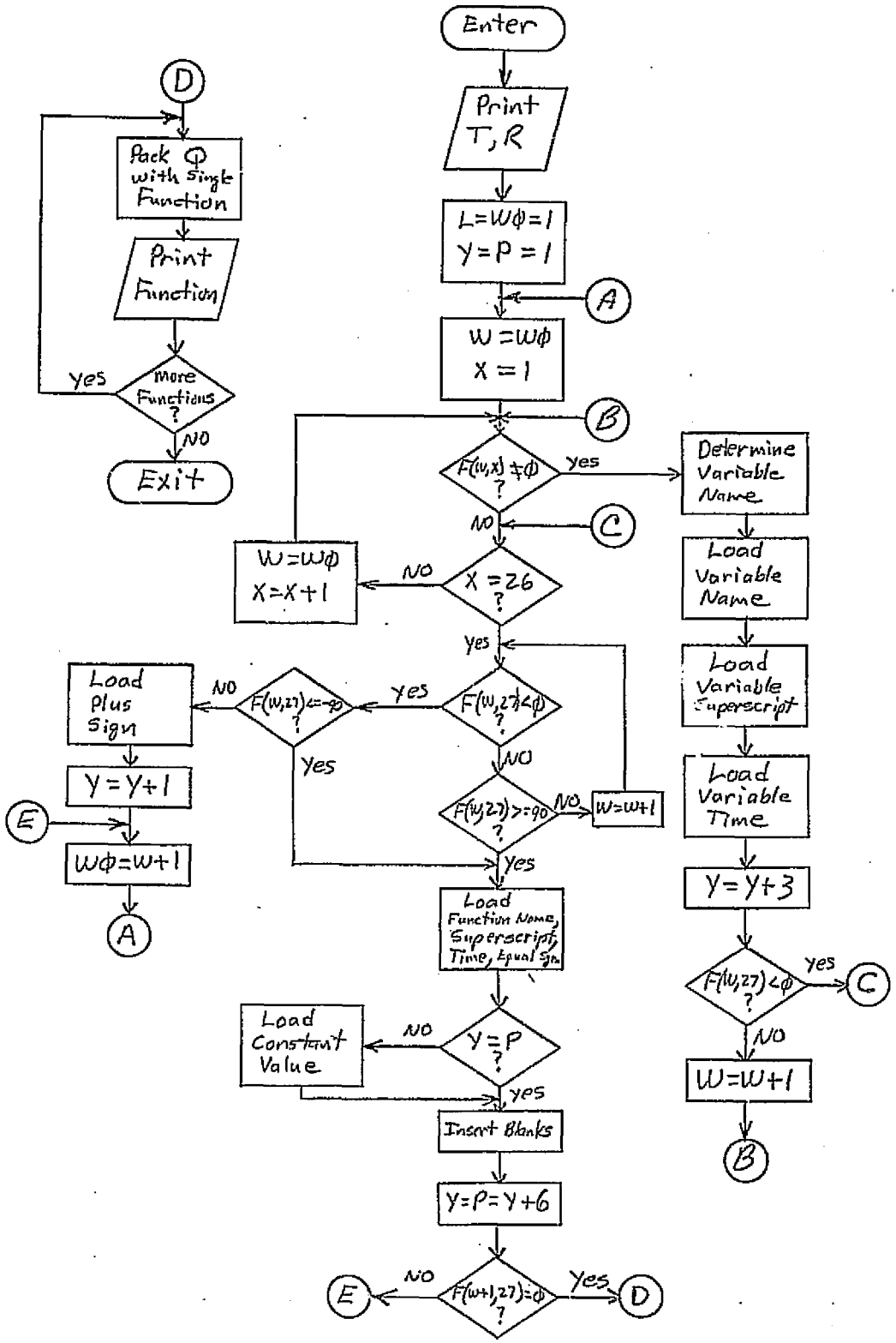


Fig A.8- Flowchart for OUTPUT subroutine.

RETRIEVE FUNCTIONS Subroutine

Functions needed during the race analysis function modification routine are retrieved from G by this subroutine. Called by FUNCTION MODIFICATION subroutine.

FUNCTION TRANSFER Subroutine

New functions produced during race analysis are placed into G by this program. Pointers in L are also updated. Called by FUNCTION MODIFICATION subroutine.

```
1 REM SIMLOG-2A
2 REM 8/14/75
3 REM PERFORMS RACE ANALYSIS.  OUTPUTS EQUATIONS TO FILE.
10 OPEN  PRINTOUT  TO:1,PRINT OVER
100 DIM F(100,27),G(100,27),H(100,27),L(26,4),N(27,27),A(10)
105 DIM M(2,5)
106 MAT M=ZER
110 DIM T(100,27)1MAT T=ZER
120 DIM P(600)1MAT P=ZER
130 DIM Q(72)
135 MAT Q=ZER
140 MAT F=ZER1MAT G=ZER1MAT H= ZER1MAT L= ZER1MAT N=ZER1MAT A=ZER
150 FOR M=1 TO 1001F(N,27)=-901NEXT M
160 MAT L=CON
170 Z9=0
180 PRINT ENTER GATE OUTPUT SYMBOL FOLLOWED BY GATE INPUT SYMBOLS.
190 PRINT NETWORK INPUT SYMBOLS SHOULD BE FOLLOWED BY A MINUS SIGN.
200 PRINT WHEN ALL GATES HAVE BEEN ENTERED--ENTER 0.
210 E=1
220 INPUT B$
225 MAT A=ZER
230 CHANGE B$ TO A
240 C=LEN(B$)
250 N(E,27)=A(1)
260 A=1
270 IF A(A)=240 GOTO 520
280 IF A(A)=96 GOTO 360
290 IF A(A)>201 GOTO 320
300 A(A)=A(A)-192
310 GOTO 360
320 IF A(A)>217 GOTO 3F
330 A(A)=A(A)-199
340 GOTO 360
350 A(A)=A(A)-207
360 A=A+1
370 IF A<=C GOTO 280
380 B=1
390 D=1
400 N(E,B)=A(D)
410 IF B>=C GOTO 500
420 B=B+1
430 D=D+1
440 IF A(D+1)=96 GOTO 470
450 N(E,B)=A(D)
460 GOTO 410
470 N(E,B)=-A(D)
480 D=D+1
490 GOTO 410
500 E=E+1
510 GOTO 220
520 REM
630 B=1
640 FOR A=1 TO 100
650 IF N(A,1)=0 GOTO 691
```

Network Entry

```
660 L(N(A+1),1)=B
670 L(N(A+1),2)=B+1
680 B=B+2
690 NEXT A
691 PRINT ENTER CROSS-COUPLED GATE PAIRS--ENTER 0 TO STOP.
692 X=11Y=1
693 INPUT B$
694 MAT A=ZER
695 CHANGE B$ TO A
696 IF A(1)=240 GOTO 710
697 IF A(Y)>201 GOTO 700
698 M(Y,X)=A(Y)-192
699 GOTO 704
700 IF A(Y)>217 GOTO 703
701 M(Y,X)=A(Y)-199
702 GOTO 704
703 M(Y,X)=A(Y)-207
704 M(Y,X+1)=A(Y)
705 IF Y=2 GOTO 707
706 Y=Y+11GOTO 697
707 Y=1
708 X=X+2
709 GOTO 693
710 PRINT ENTER STARTING STATE(0 OR 1) FOR EACH SIGNAL LINE BY
720 PRINT TYPING THE LINE SYMBOL FOLLOWED BY =0 OR =1.
730 PRINT TO TERMINATE ENTER J.
740 PRINT SIGNAL LINES NOT ENTERED ARE TREATED AS UNKNOWN CONDITIONS.
750 MAT A=ZER
760 REM
770 INPUT B$
780 CHANGE B$ TO A
790 A(2)=A(1)
800 IF A(1)=240 GOTO 960
810 A(3)=A(3)-240
820 IF A(1)>201 GOTO 850
830 A(1)=A(1)-192
840 GOTO 890
850 IF A(1)>217 GOTO 880
860 A(1)=A(1)-199
870 GOTO 890
880 A(1)=A(1)-207
890 IF A(3)=1 GOTO 930
900 F(L(A(1),1),27)=-A(2)
910 F(L(A(1),2),27)=A(2)
920 GOTO 760
930 F(L(A(1),1),27)=A(2)
940 F(L(A(1),2),27)=-A(2)
950 GOTO 760
960 I=11R=1
970 I=11J=01H=11S=1
980 MAT H=ZER
1000 P=11K=11H=1
1040 IF N(S,P)=0 GOTO 2090
1050 P=P+1
1060 IF N(S,P)=0 GOTO 1340
```

Enter Cross-Coupled NAND Pairs

Starting State Entry

```
1070 IF N(S,P)<0 GOTO 1200
1090 FOR A=L(N(S,P),I+1) TO 100
1100 FOR M=1 TO 27
1110 H(H,M)=F(A,M)
1120 NEXT M
1130 H=H+1
1140 IF F(A,27)<=-90 GOTO 1170
1150 IF F(A,27)>=90 GOTO 1170
1160 NEXT A
1170 REM
1190 GOTO 1260
1200 IF I=0 GOTO 1240
1210 H(H,-N(S,P))=-T
1230 GOTO 1250
1240 H(H,-N(S,P))=T
1250 H(H,27)=-N(S,27)1H=H+1
1260 REM
1300 IF K=0 GOTO 1340
1310 REM
1320 K=0
1330 GOTO 1050
1340 IF I=1 GOTO 1400
1370 GOSUB 2530
1390 GOTO 1430
1400 REM
1410 GOSUB 4070
1430 IF N(S,P+1)<>0 GOTO 1050
1460 A1=L(N(S,1),J+3)
1490 MAT T=ZER1MAT T=G
1560 FOR A=1TO H-1
1570 REM
1580 FOR M=1 TO 27
1590 G(A1,M)=H(A,M)
1600 NEXT M
1610 A1=A1+11NEXT A
1620 IF J=0 GOTO 1660
1630 IF N(S+1,1)=0 GOTO 1780
1640 IF L(N(S,1),4)>=L(N(S+1,1),3)GOTO17701A2=L(N(S+1,1),3)
1650 GOTO 1670
1660 IF L(N(S,1),3)>=L(N(S,1),4)GoTO 17701A2=L(N(S,1),4)
1670 FOR M=A1 TO 100
1680 FOR M1=1 TO 27
1690 G(M,M1)=0
1700 NEXT M1
1710 NEXT M
1720 FOR M=A1 TO 100
1730 FOR M1=1 TO 27
1740 G(M,M1)=T(A2,M1)
1750 IF A2=100 GOTO 1770
1760 NEXT M11A2=A2+11NEXT M
1770 GOTO 1860
1780 FOR M=A1 TO 100
1790 FOR M1=1 TO 27
1800 G(M,M1)=0
1810 NEXT M1
```

Load H with Function from F

Copy H into G

```
1820 NEXT M
1860 MAT H=ZER
1870 IF I=0 GOTO 1910
1880 D=L(N(S,1),3)+H-1-L(N(S,1),4)
1890 L(N(S,1),4)=L(N(S,1),3)+H-1
1900 GOTO 1980
1910 IF N(S+1,1)<>0 GOTO 1940
1920 S=S+1
1930 GOTO 2010
1940 D=L(N(S,1),4)+H-1-L(N(S+1,1),3)
1950 L(N(S+1,1),3)=L(N(S,1),4)+H-1
1960 L(N(S+1,1),4)=L(N(S+1,1),4)+D
1970 S=S+1
1980 FOR X=S+1 TO 261IF N(X,1)=0 GOTO 2000
1990 L(N(X,1),3)=L(N(X,1),3)+DL(N(X,1),4)=L(N(X,1),4)+D
2000 NEXT X
2010 IF I=0 GOTO 2050
2020 REM
2030 I=0IJ=1
2040 GOTO 1000
2050 REM
2060 I=1IJ=0
2080 GOTO 1000
2090 REM
2180 IF G(M,27)<=-90 GOTO 2210
2210 REM
2220 FOR M= 1 TO 100
2230 FOR M1=1 TO 27
2240 IF F(M,M1)<>G(M,M1) GOTO 2286
2250 NEXT M1
2270 NEXT M
2280 Z9=1
2285 GOTO 2340
2286 IF M(1,1)=0 GOTO 2340
2287 IF R=1 GOTO 2340
2288 PRINT
2289 PRINT BEGIN RACE ANALYSIS.
2290 PRINT
2291 E=1
2292 IF M(1,E)=0 GOTO 2340
2293 E1=L(M(1,E),2)
2294 E2=L(M(1,E),4)
2295 L=0
2296 GOSUB 7000
2298 IF E3=1 GOTO 2312
2300 E1=L(M(2,E),1)
2301 L=1
2302 E2=L(M(2,E),3)
2304 GOSUB 7000
2306 IF E3=1 GOTO 2312
2308 GOSUB 9000
2310 GOTO 2330
2312 E1=L(M(2,E),2)
2314 E2=L(M(2,E),4)
2315 L=0
```

Update L Matrix

Check Equality of Matrix F and Matrix G

Perform Race Analysis

```
2316 GOSUB 7000
2318 IF E3=1 GOTO 2330
2320 E1=L(M(1,E),1)
2322 E2=L(M(1,E),3)
2323 L=1
2324 GOSUB 7000
2326 IF E3=1 GOTO 2330
2328 GOSUB 9000
2330 E=E+2
2332 GOTO 2292
2340 MAT F=ZER
2350 FOR M=1 TO 100
2360 FOR M1=1 TO 27
2370 F(M,M1)=G(M,M1)
2380 IF M1=27 GOTO 2400
2390 L(M1,1)=L(M1,3)1L(M1,2)=L(M1,4)
2400 NEXT M1
2410 NEXT M
2420 GOTO 2490
2430 PRINT  NEXT T ;
2435 C$=
2440 INPUT C$
2450 IF C$= NO  GOTO 6420
2460 T=T+1
2470 Z9=0
2480 GOTO 2510
2490 GOSUB 5710
2500 IF Z9=1 GOTO 2430
2510 R=R+1
2520 GOTO 970
2530 REM AND1
2535 REM
2540 X1=1
2550 IF H(X1,27)<=-90 GOTO 2590
2560 IF H(X1,27)>=90 GOTO 2590
2570 X1=X1+1
2580 GOTO 2550
2590 Z3=X1+11X3=Z3
2600 X1=X1+1
2605 IF H(X1,27)=0 GOTO 4040
2610 IF H(X1,27)<=-90 GOTO 2640
2620 IF H(X1,27)>=90 GOTO 2640
2630 GOTO 2600
2640 Y=1
2650 X6=X1+1
2660 Z5=X1+11X5=Z5
2670 IF H(1,27)>=90 GOTO 2700
2680 IF H(Z3,27)>=90 GOTO 3810
2690 GOTO 2720
2700 IF H(Z3,27)>=90 GOTO 3880
2710 GOTO 3820
2720 IF H(1,Y)<>0 GOTO 2760
2730 IF Y=26 GOTO 3790
2740 Y=Y+1
2750 GOTO 2720
```

Update F and L

Locate End of First Function

Locate End of Second Function

Check for Identity Functions

Check for Zero Functions

```
2760 Y=1
2770 IF H(X3,Y)<>0 GOTO 2800
2780 IF Y=26 GOTO 3790
2790 Y=Y+1:GOTO 2770
2800 Y=1
2810 X1=1
2820 X2=X1
2830 X4=X3
2840 X6=X3
2850 IF H(X2,Y)=H(X4,Y) GOTO 2930
2860 IF H(X4,27)<0 GOTO 2890
2870 X4=X4+1
2880 GOTO 2850
2890 IF H(X2,27)<0 GOTO 2950
2900 X4=X3
2910 X2=X2+1
2920 GOTO 2850
2930 IF H(X2,Y)=0 GOTO 2860
2940 GOTO 3290
2950 X0=X1
2960 H(X6,Y)=H(X0,Y)
2970 H(X6,27)=1
2980 IF X0<X2 GOTO 3010
2990 X0=X3
3000 GOTO 3040
3010 X0=X0+1
3020 X6=X6+1
3030 GOTO 2960
3040 X6=X6+1
3050 REM
3060 H(X6,Y)=H(X0,Y)
3070 H(X6,27)=1
3080 IF X0<X4 GOTO 3110
3090 H(X6,27)=-1
3100 GOTO 3130
3110 X0=X0+1
3120 GOTO 3040
3130 X7=X5
3140 X8=X5+1
3150 IF H(X7,Y)<>0 GOTO 3180
3160 H(X7,Y)=H(X8,Y)
3170 H(X8,Y)=0
3180 IF H(X7,Y)=H(X8,Y) GOTO 3310
3190 IF H(X8,27)<0 GOTO 3220
3200 X8=X8+1
3210 GOTO 3150
3220 IF X8=X7+1 GOTO 3260
3230 X7=X7+1
3240 X8=X7+1
3250 GOTO 3150
3260 IF Y<26 GOTO 3280
3270 GOTO 3620
3280 Y=Y+1
3290 REM
3300 GOTO 2820
```

Select Terms To Be ANDed

Check for Zero Product

Merge Terms

Look for duplicate variables.

Select next column.

```
3310 IF H(X7,Y)=0 GOTO 3190
3320 X0=X8
3330 IF X0=X6 GOTO 3370
3340 H(X0,Y)=H(X0+1,Y)
3350 X0=X0+1
3360 GOTO 3330
3370 H(X0,Y)=0
3380 GOTO 3190
3390 Y=1
3400 REM
3410 H(X6,Y)=0
3420 IF Y<26 GOTO 3450
3430 H(X6,27)=-1
3440 GOTO 3470
3450 Y=Y+1
3460 GOTO 3420
3470 IF H(X4,27)<0 GOTO 3500
3480 X4=X4+1
3490 GOTO 3470
3500 IF X4=Z5-1 GOTO 3550
3510 X3=X4+1
3520 X5=X6+1
3530 Y=1
3540 GOTO 2820
3550 IF H(X2,27)<0 GOTO 3580
3560 X2=X2+1
3570 GOTO 3550
3580 IF X2=Z3-1 GOTO 3790
3590 X1=X2+1
3600 X3=Z3
3610 GOTO 3620
3620 X5=X5
3630 IF H(X6,27)<0 GOTO 3660
3640 X6=X6+1
3650 GOTO 3630
3660 Y=1
3670 IF H(X6,Y)=0 GOTO 3700
3680 H(X6,27)=-1
3690 GOTO 3470
3700 IF Y<26 GOTO 3770
3710 IF X6=X5 GOTO 3750
3720 H(X6,27)=0
3730 X6=X6-1
3740 GOTO 3660
3750 H(X6,27)=0
3760 GOTO 3470
3770 Y=Y+1
3780 GOTO 3720
3790 H(X6,27)=-N(5,27)
3800 GOTO 3910
3810 X3=11GOTO 3830
3820 X3=Z3
3830 FOR X5=Z5 TO 100
3840 FOR Y=1 TO 271H(X5,Y)=H(X3,Y)1NEXT Y
3850 IF H(X3,27)<=-90 GOTO 3910
```

Remove duplicate variables.

Enter all zero term.

Select next two terms to be ANDed, if any.

Find and remove zero rows in non-zero product term.

Copy Non-identity function as product.

```
3860 X3=X3+1:NEXT X5
3870 GOTO 3410
3880 H(1,27)=N(S,27):H=2
3890 FOR A=2 TO 100
3891 FOR A1=1 TO 27:H(A,A1)=0:NEXT A1
3892 NEXT A
3900 GOTO 4060
3910 C1=0
3920 Z4=Z5
3930 FOR A=1 TO 100-Z5
3935 C1=C1+1
3940 FOR A1=1 TO 27
3950 H(A,A1)=H(Z4,A1)
3960 NEXT A1
3970 Z4=Z4+1
3975 IF H(Z4-1,27)<=-90 GOTO 3990
3980 NEXT A
3990 FOR A=C1+1 TO 100
4000 FOR A1=1 TO 27
4010 H(A,A1)=0
4020 NEXT A1
4030 NEXT A
4040 GOSUB 4270
4050 H=X1+1
4060 RETURN
4070 REM GR- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
4075 REM
4080 X=1
4090 IF H(X,27)<=-90 GOTO 4130
4100 IF H(X,27)>=90 GOTO 4210
4110 IF X<100 GOTO 4160
4120 PRINT ERROR 1:GOTO 4180
4130 IF X<100 GOTO 4140:GOTO 4180
4140 IF H(X+1,27)=0 GOTO 4180
4150 H(X,27)=-1
4160 X=X+1
4170 GOTO 4090
4180 GOSUB 4270
4190 H=X1+1
4200 GOTO 4260
4210 MAT H=ZER
4220 H(1,27)=N(S,27)
4230 H=2
4260 RETURN
4270 REM MINIMIZATION ROUTINNE - - - - - - - - - - - - - - - - - - - - -
4280 X1=1
4290 IF H(X1,27)<=-90 GOTO 5300
4292 IF H(X1,27)<0 GOTO 4300
4294 X1=X1+1
4296 GOTO 4290
4300 A1=1
4310 A2=1
4320 IF H(X2,27)<0 GOTO 4350
4330 A2=X2+1
4340 GOTO 4320
```

zero terms 2 to 100 of H array.

Move product to top of H array. Zero bottom rows of H array

Check for single term function.

Locate second term.

```
4350 X2=X2+1
4370 W1=1
4375 FOR X=X1 TO 100
4377 X3=X
4380 FOR Y=1 TO 26
4390 IF H(X,Y)<>0 GOTO 4440
4400 NEXT Y
4410 IF H(X,27)<0 GOTO 4430
4420 NEXT X
4430 W1=0
4440 X2=1
4445 FOR X=X2 TO 100
4447 X4=X
4450 FOR Y=1 TO 26
4460 IF H(X,Y)<>0 GOTO 4501
4470 NEXT Y
4480 IF H(X,27)<0 GOTO 4500
4490 NEXT X
4500 W2=0
4501 IF W1=0 GOTO 5070
4502 IF W2=0 GOTO 5070
4510 Z=0
4520 Y=1
4530 X3=X1
4540 X4=X2
4550 IF H(X3,Y)=0 GOTO 4660
4560 IF H(X4,Y)<>0 GOTO 4690
4570 IF Z=2 GOTO 5070
4580 Z=1
4590 GOTO 4660
4600 IF H(X4,Y)=0 GOTO 4640
4610 IF Z=1 GOTO 5070
4620 Z=2
4630 GOTO 4660
4640 IF Z<>0 GOTO 4660
4650 Z=3
4660 REM
4665 IF Y>=26 GOTO 5180
4670 Y=Y+1
4680 GOTO 4530
4690 C=0
4700 IF H(X3,Y)<>H(X4,Y) GOTO 4740
4710 C1=0
4720 C=C+1
4730 GOTO 4750
4740 C1=1
4750 IF H(X4,27)<0 GOTO 4830
4760 IF H(X4+1,Y)=0 GOTO 4830
4770 IF H(X3,27)<0 GOTO 4810
4780 IF H(X3+1,Y)=0 GOTO 4810
4790 IF C1=1 GOTO 4810
4800 X3=X3+1
4810 X4=X4+1
4820 GOTO 4700
4830 IF H(X3,27)<0 GOTO 4880
```

Check for zero terms.

Initialize for comparison

Check terms for presence of variables.

Column selection.

Count number of common variables.

```
4840 IF H(X3+1,Y)=0 GOTO 4880
4850 X4=X3
4860 X3=X3+1
4870 GOTO 4700
4880 IF H(X3,Y)<>0 GOTO 4900
4890 X3=X3-1
4900 IF H(X4,Y)<>0 GOTO 4920
4910 X4=X4-1
4920 IF C=X3-X1+1 GOTO 4930
4930 IF C=X4-X2+1 GOTO 4950
4940 GOTO 5070
4950 REM
4960 IF Z=2 GOTO 5070
4970 Z=1
4980 GOTO 4660
4990 IF C=X4-X2+1 GOTO 5040
5010 IF Z=1 GOTO 5070
5020 Z=2
5030 GOTO 4660
5040 IF Z<>0 GOTO 4660
5050 Z=3
5060 GOTO 4660
5070 REM
5071 IF H(X4,27)<0 GOTO 5075
5072 X4=X4+1
5073 GOTO 5071
5075 IF H(X4,27)<=-90 GOTO 5100
5080 X2=X4+1
5090 GOTO 4370
5100 IF H(X3,27)<0 GOTO 5105
5101 X3=X3+1
5102 GOTO 5100
5105 IF X2=X3+1 GOTO 5300
5110 X1=X3+1
5120 X2=X1
5130 IF H(X2,27)<0 GOTO 5160
5140 X2=X2+1
5150 GOTO 5130
5160 X2=X2+1
5170 GOTO 4370
5180 IF Z=0 GOTO 5070
5185 IF Z=2 GOTO 5260
5190 IF H(X3,27)<0 GOTO 5220
5200 X3=X3+1
5210 GOTO 5190
5220 FOR X=X1 TO X3
5230 FOR Y=1 TO 26
5240 H(X,Y)=0
5250 NEXT Y
5251 NEXT X
5252 IF H(X4,27)<0 GOTO 5070
5253 X4=X4+1
5254 GOTO 5252
5260 IF H(X4,27)<0 GOTO 5270
5265 X4=X4+1
```

Check for coverage or equality.

Select next terms for comparison.

Replace redundant term with zero term.

```
5266 GOTO 5260
5270 FOR X=X2 TO X4
5275 FOR Y=1 TO 26
5280 H(X,Y)=0
5285 NEXT Y
5290 NEXT X
5291 IF H(X3,27)<0 GOTO 5070
5292 X3=X3+1
5293 GOTO 5291
5300 X0=1
5310 X1=X0
5320 Y=1
5330 IF Y=27 GOTO 5430
5340 IF H(X1,Y)=0 GOTO 5410
5350 IF H(X1,27)<=-90 GOTO 5560
5360 IF H(X1,27)<0 GOTO 5390
5370 X1=X1+1
5380 GOTO 5350
5390 X0=X1+1
5400 GOTO 5310
5410 Y=Y+1
5420 GOTO 5330
5430 IF H(X1,27)<0 GOTO 5460
5440 X1=X1+1
5450 GOTO 5320
5460 IF H(X1,27)<=-90 GOTO 5620
5470 Z0=0
5480 FOR Z=X1+1 TO 100
5490 FOR Z1=1 TO 27
5500 H(X0+Z0,Z1)=H(Z,Z1)
5510 NEXT Z1
5520 IF H(Z,27)<=-90 GOTO 5300
5530 Z0=Z0+1
5540 NEXT Z
5550 GOTO 5300
5560 FOR Z2=X1+1 TO 100
5570 FOR Z1=1 TO 27
5580 H(Z2,Z1)=0
5590 NEXT Z1
5600 NEXT Z2
5610 GOTO 5680
5620 IF X1=1 GOTO 5660
5630 X1=X1-1
5640 IF H(X1,27)<0 GOTO 5560
5650 GOTO 5630
5660 H(X1,27)=-N(5,27)
5670 GOTO 5560
5680 H(X1,27)=-N(5,27)
5685 REM
5690 RETURN
5700 REM
5710 REM OUTPUT ROUTINE- - - - - - - - - - - - - - - - -
5715 MAT P=ZER
5717 PRINT
5720 PRINT T= ;T, R= ;R1PRINT
```

Eliminate zero terms and pack non-zero terms in top of H array.

Label function.

ORIGINAL PAGE IS OF POOR QUALITY

```
5725 PRINT:1, T= ;T, R= ;R
5730 L=1
5740 W0=11Y=11P=1
5750 W=W0
5760 X=1
5770 IF F(W,X)<>0 GOTO 5820
5780 IF X=26 GOTO 5850
5790 W=W0
5800 X=X+1
5810 GOTO 5770
5820 IF X<= 9 GOTO 6150
5830 IF X<=18 GOTO 6150
5840 IF X<=26 GOTO 6170
5850 IF F(W,27)<0 GOTO 5910
5860 IF F(W,27)>=90 GOTO 5950
5870 W=W+1
5880 GOTO 5850
5890 W0=W+1
5900 GOTO 5750
5910 IF F(W,27)<=-90 GOTO 5950
5920 P(Y+4)=76
5930 Y=Y+1
5940 GOTO 5990
5950 P(P)=ABS(F(W,27))
5960 P(P+2)=T+240
5970 P(P+1)=L+240
5980 L=1-L
5990 P(P+3)=126
6000 IF Y<>P GOTO 6070
6010 REM
6020 IF F(W,27)>0 GOTO 6040
6030 P(Y+4)=240
6040 GOTO 6060
6050 P(Y+4)=241
6060 Y=Y+1
6070 P(Y+4)=64
6080 P(Y+5)=64
6090 P=Y+6
6100 Y=P
6110 IF F(W+1,27)=0 GOTO 6280
6120 GOTO 5990
6130 P(Y+4)=X+192
6140 GOTO 6180
6150 P(Y+4)=X+199
6160 GOTO 6180
6170 P(Y+4)=X+217
6180 IF F(W,X)<0 GOTO 6210
6190 P(Y+5)=241
6200 GOTO 6220
6210 P(Y+5)=240
6220 P(Y+6)=ABS(F(W,X))+240
6230 REM
6240 Y=Y+3
6250 IF F(W,27)<0 GOTO 5780
6260 W=W+1
```

ORIGINAL PAGE IS
OF POOR QUALITY

Determine variable name.

End of term or end of function?

Identity function?

End of function?

Load plus char.

Load function name, superscript, time, and equal sign.

Check for constant function.

Load constant value.

Load blanks.

Packing complete?

Load variable name.

Load variable superscript.

Load variable time.

```
6270 GOTO 6770
6280 MAT G = ZER
6290 M=1
6300 M1=1
6310 G(M1)=P(M)
6320 IF P(M)<>64 GOTO 6380
6330 M1=0
6340 CHANGE Q TO R$
6345 PRINT :1,R$
6350 PRINT R$
6360 MAT G=ZER
6370 IF P(M+1)=0 GOTO 6410
6380 M=M+1
6390 M1=M1+1
6400 GOTO 6310
6410 RETURN
6420 GOTO 6700
6430 REM H MATRIX OUTPUT - - - - - - - - - - - - - - - - - - - - - - - -
6440 PRINT H MATRIX
6450 FOR A=1 TO 100
6460 IF H(A,27)=0 GOTO 6500
6470 FOR A1=1 TO 27
6480 PRINT:1,H(A,A1);
6490 NEXT A1
6491 PRINT
6492 NEXT A
6500 PRINT
6510 PRINT
6520 RETURN
6700 GOTO 10860
7000 REM SUBROUTINE--FUNCTION IDENTITY CHECK - - - - - - - - - - - - - -
7005 MAT H=ZER
7010 E4=1
7050 E5=E1
7060 FOR E6=1 TO 100
7070 FOR E7=1 TO 27
7080 H(E6,E7)=F(E5,E7)
7090 NEXT E7
7100 IF F(E5,27)<=-90 GOTO 7140
7110 IF F(E5,27)>=90 GOTO 7140
7120 E5=E5+1
7130 NEXT E6
7140 GOSUB 7500
7142 MAT H=ZER
7145 E4=0
7150 E5=E2
7160 FOR E6=1 TO 100
7170 FOR E7=1 TO 27
7180 H(E6,E7)=G(E5,E7)
7190 NEXT E7
7200 IF G(E5,27)<=-90 GOTO 7240
7210 IF G(E5,27)>=90 GOTO 7240
7220 E5=E5+1
7230 NEXT E6
7240 GOSUB 7500
```

Pack Q and print.

```
4840 IF H(X3+1,Y)=0 GOTO 4880
4850 X4=X2
4860 X3=X3+1
4870 GOTO 4700
4880 IF H(X3,Y)<>0 GOTO 4900
4890 X3=X3-1
4900 IF H(X4,Y)<>0 GOTO 4920
4910 X4=X4+1
4920 IF C=X3-X1+1 GOTO 4990
4930 IF C=X4-X2+1 GOTO 4950
4940 GOTO 5070
4950 REM
4960 IF Z=2 GOTO 5070
4970 Z=1
4980 GOTO 4660
4990 IF C=X4-X2+1 GOTO 5040
5010 IF Z=1 GOTO 5070
5020 Z=2
5030 GOTO 4660
5040 IF Z<>0 GOTO 4660
5050 Z=3
5060 GOTO 4660
5070 REM
5071 IF H(X4,27)<0 GOTO 5075
5072 X4=X4+1
5073 GOTO 5071
5075 IF H(X4,27)<=-90 GOTO 5100
5080 X2=X4+1
5090 GOTO 4370
5100 IF H(X3,27)<0 GOTO 5105
5101 X3=X3+1
5102 GOTO 5100
5105 IF X2=X3+1 GOTO 5300
5110 X1=X3+1
5120 X2=X1
5130 IF H(X2,27)<0 GOTO 5160
5140 X2=X2+1
5150 GOTO 5130
5160 X2=X2+1
5170 GOTO 4370
5180 IF Z=0 GOTO 5070
5185 IF Z=2 GOTO 5260
5190 IF H(X3,27)<0 GOTO 5220
5200 X3=X3+1
5210 GOTO 5190
5220 FOR X=X1 TO X3
5230 FOR Y=1 TO 26
5240 H(X,Y)=0
5250 NEXT Y
5251 NEXT X
5252 IF H(X4,27)<0 GOTO 5070
5253 X4=X4+1
5254 GOTO 5252
5260 IF H(X4,27)<0 GOTO 5270
5265 X4=X4+1
```

Check for coverage or equality.

Select next terms for comparison.

Replace redundant term with zero term.

```
5266 GOTO 5260
5270 FOR X=X2 TO X4
5275 FOR Y=1 TO 26
5280 H(X,Y)=0
5285 NEXT Y
5290 NEXT X
5291 IF H(X3,27)<0 GOTO 5070
5292 X3=X3+1
5293 GOTO 5291
5300 X0=1
5310 X1=X0
5320 Y=1
5330 IF Y=27 GOTO 5430
5340 IF H(X1,Y)=0 GOTO 5410
5350 IF H(X1,27)<=-90 GOTO 5560
5360 IF H(X1,27)<0 GOTO 5390
5370 X1=X1+1
5380 GOTO 5350
5390 X0=X1+1
5400 GOTO 5310
5410 Y=Y+1
5420 GOTO 5330
5430 IF H(X1,27)<0 GOTO 5460
5440 X1=X1+1
5450 GOTO 5320
5460 IF H(X1,27)<=-90 GOTO 5620
5470 Z0=0
5480 FOR Z=X1+1 TO 100
5490 FOR Z1=1 TO 27
5500 H(X0+Z0,Z1)=H(Z,Z1)
5510 NEXT Z1
5520 IF H(Z,27)<=-90 GOTO 5300
5530 Z0=Z0+1
5540 NEXT Z
5550 GOTO 5300
5560 FOR Z2=X1+1 TO 100
5570 FOR Z1=1 TO 27
5580 H(Z2,Z1)=0
5590 NEXT Z1
5600 NEXT Z2
5610 GOTO 5680
5620 IF X1=1 GOTO 5660
5630 X1=X1-1
5640 IF H(X1,27)<0 GOTO 5560
5650 GOTO 5630
5660 H(X1,27)=-N(S,27)
5670 GOTO 5560
5680 H(X1,27)=-N(S,27)
5685 REM
5690 RETURN
5700 REM
5710 REM OUTPUT ROUTINE- - - - - - - - - - - - - - - - - - - - - - - - - - - -
5715 MAT P=ZER
5717 PRINT
5720 PRINT T= ;T, R= ;R1PRINT
```

Eliminate zero terms and pack non-zero terms in top of H array.

Label function.

```
5725 PRINT:1, T= ;T, R= ;R
5730 L=1
5740 W0=1:Y=11:P=1
5750 W=W0
5760 X=1
5770 IF F(W,X)<>0 GOTO 5820
5780 IF X=26 GOTO 5850
5790 W=W0
5800 X=X+1
5810 GOTO 5770
5820 IF X<= 9 GOTO 6130
5830 IF X<=18 GOTO 6150
5840 IF X<=26 GOTO 6170
5850 IF F(W,27)<0 GOTO 5910
5860 IF F(W,27)>=90 GOTO 5950
5870 W=W+1
5880 GOTO 5850
5890 W0=W+1
5900 GOTO 5750
5910 IF F(W,27)<=-90 GOTO 5950
5920 P(Y+4)=78
5930 Y=Y+1
5940 GOTO 5890
5950 P(P)=ABS(F(W,27))
5960 P(P+2)=T+240
5970 P(P+1)=L+240
5980 L=1-L
5990 P(P+3)=126
6000 IF Y<>P GOTO 6070
6010 REM
6020 IF F(W,27)>0 GOTO 6050
6030 P(Y+4)=240
6040 GOTO 6060
6050 P(Y+4)=241
6060 Y=Y+1
6070 P(Y+4)=64
6080 P(Y+5)=64
6090 P=Y+6
6100 Y=P
6110 IF F(W+1,27)=0 GOTO 6280
6120 GOTO 5890
6130 P(Y+4)=X+192
6140 GOTO 6180
6150 P(Y+4)=X+199
6160 GOTO 6180
6170 P(Y+4)=X+217
6180 IF F(W,X)<0 GOTO 6210
6190 P(Y+5)=241
6200 GOTO 6220
6210 P(Y+5)=240
6220 P(Y+6)=ABS(F(W,X))+240
6230 REM
6240 Y=Y+3
6250 IF F(W,27)<0 GOTO 5780
6260 W=W+1
```

ORIGINAL PAGE IS
OF POOR QUALITY

Determine variable name.

End of term or end of function?

Identity function?

End of function?

Load plus char.

Load function name, superscript, time, and equal sign.

Check for constant function.

Load constant value.

Load blanks.

Packing complete?

Load variable name.

Load variable superscript.

Load variable time.

```
6270 GOTO 5770
6280 MAT G = ZER
6290 M=1
6300 M1=1
6310 G(M1)=P(M)
6320 IF P(M)<>64 GOTO 6380
6330 M1=0
6340 CHANGE Q TO R$
6345 PRINT #1,R$
6350 PRINT R$
6360 MAT Q=ZER
6370 IF P(M+1)=0 GOTO 6410
6380 M=M+1
6390 M1=M1+1
6400 GOTO 6310
6410 RETURN
6420 GOTO 6700
6430 REM H MATRIX OUTPUT - - - - - - - - - - - - - - - - - - - - - - - - - -
6440 PRINT H MATRIX
6450 FOR A=1 TO 100
6460 IF H(A,27)=0 GOTO 6500
6470 FOR A1=1 TO 27
6480 PRINT#1,H(A,A1);
6490 NEXT A1
6491 PRINT
6492 NEXT A
6500 PRINT
6510 PRINT
6520 RETURN
6700 GOTO 10860
7000 REM SUBROUTINE--FUNCTION IDENTITY CHECK - - - - - - - - - - - - - - -
7005 MAT H=ZER
7010 E4=1
7050 E5=E1
7060 FOR E6=1 TO 100
7070 FOR E7=1 TO 27
7080 H(E6,E7)=F(E5,E7)
7090 NEXT E7
7100 IF F(E5,27)<=-90 GOTO 7140
7110 IF F(E5,27)>=90 GOTO 7140
7120 E5=E5+1
7130 NEXT E6
7140 GOSUB 7500
7142 MAT H=ZER
7145 E4=0
7150 E5=E2
7160 FOR E6=1 TO 100
7170 FOR E7=1 TO 27
7180 H(E6,E7)=G(E5,E7)
7190 NEXT E7
7200 IF G(E5,27)<=-90 GOTO 7240
7210 IF G(E5,27)>=90 GOTO 7240
7220 E5=E5+1
7230 NEXT E6
7240 GOSUB 7500
```

Pack Q and print.

```
7245 B$=
7250 PRINT FUNCTIONS IDENTICAL ;
7260 INPUT B$
7265 PRINT
7270 IF B$= YES  GOTO 7300
7280 E3=0
7290 GOTO 7310
7300 E3=1
7310 RETURN
7500 REM SUBROUTINE -FUNCTION IDENTITY CHECK-OUTPUT - - - - - - - - - - - - -
7510 MAT P=ZER
7540 W0=11Y=11P=1
7550 W=W0
7560 X=1
7570 IF H(W,X)<>0 GOTO 7620
7580 IF X=26 GOTO 7650
7590 W=W0
7600 X=X+1
7610 GOTO 7570
7620 IF X<= 9 GOTO 7930
7630 IF X<=13 GOTO 7950
7640 IF X<=26 GOTO 7970
7650 IF H(W,27)<0 GOTO 7710
7660 IF H(W,27)>=90 GOTO 7750
7670 W=W+1
7680 GOTO 7650
7690 W0=W+1
7700 GOTO 7550
7710 IF H(W,27)<=-90 GOTO 7750
7720 P(Y+4)=78
7730 Y=Y+1
7740 GOTO 7690
7750 P(P)=ABS(H(W,27))
7760 P(P+2)=75
7770 P(P+1)=L+240
7790 P(P+3)=126
7800 IF Y<>P GOTO 7570
7810 P(Y+4)=64
7820 IF H(W,27)>0 GOTO 7850
7830 P(Y+5)=240
7840 GOTO 7860
7850 P(Y+5)=241
7860 Y=Y+2
7870 P(Y+4)=32
7880 P(Y+5)=41
7890 P=Y+6
7900 Y=P
7910 IF H(W+1,27)=0 GOTO 8080
7920 GOTO 7690
7930 P(Y+4)=X+192
7940 GOTO 7980
7950 P(Y+4)=X+199
7960 GOTO 7980
7970 P(Y+4)=X+207
7980 IF H(W,X)<0 GOTO 8010
```

C-2

```
7990 P(Y+5)=241
8000 GOTO 8020
8010 P(Y+5)=240
8020 P(Y+6)=ABS(H(N,X))+240
8030 REM
8040 Y=Y+3
8050 IF H(N,27)<0 GOTO 7580
8060 N=N+1
8070 GOTO 7570
8080 MAT G = ZER
8090 M=1
8100 M1=1
8110 G(M1)=P(M)
8120 IF P(M)<>241 GOTO 8180
8130 M1=0
8140 CHANGE Q TO R$
8150 PRINT R$
8160 MAT G=ZER
8170 IF P(M+1)=0 GOTO 8210
8180 M=M+1
8190 M1=M1+1
8200 GOTO 8110
8210 RETURN
9000 REM FUNCTION MODIFICATION SUBROUTINE
9005 MO=1
9010 N1=L(M(1,E),4)
9020 FOR X=1 TO 26
9030 IF G(N1,X)<>0 GOTO 9060
9040 NEXT X
9050 GOTO 9310
9060 N2=L(M(2,E),3)
9070 FOR X=1 TO 26
9080 IF G(N2,X)<> 0 GOTO 9110
9090 NEXT X
9095 M=1
9100 GOTO 9430
9110 M=2
9120 GOTO 9430
9130 REM
9310 MO=2
9320 N3=L(M(2,E),4)
9330 FOR X=1 TO 26
9340 IF G(N3,X)<>0 GOTO 9370
9350 NEXT X
9360 GOTO 9545
9370 N4=L(M(1,E),3)
9380 FOR X=1 TO 26
9390 IF G(N4,X)<>0 GOTO 9420
9400 NEXT X
9405 M=1
9410 GOTO 9430
9420 M=2
9430 REM
9440 IF MO=2 GOTO 9480
9450 M1=N1
```

```
9460 M2=N2
9470 GOTO 9500
9480 M1=N3
9490 M2=N4
9500 GOSUB 10000
9505 IF M=1 GOTO 9520
9510 GOSUB 2530
9512 IF MO=2 GOTO 9517
9515 H(H-1,27)=-M(1,E+1)
9516 GOTO 9518
9517 H(H-1,27)=-M(2,E+1)
9518 REM
9520 IF MO=2 GOTO 9528
9521 IF M=2,GOTO 9525
9522 A1=N2
9523 NO=M(2,E)
9524 GOTO 9534
9525 A1=N1
9526 NO=M(1,E)
9527 GOTO 9534
9528 IF M=2 GOTO 9532
9529 A1=N4
9530 NO=M(1,E)
9531 GOTO 9534
9532 A1=N3
9533 NO=M(2,E)
9534 IF M=2 GOTO 9539
9535 IF MO=2 GOTO 9538
9536 H(H1-1,27)=-M(2,E+1)
9537 GOTO 9539
9538 H(H1-1,27)=-M(1,E+1)
9539 GOSUB 10500
9540 IF MO=1 GOTO 9310
9545 RETURN
9910 REM
10000 REM RETRIEVE FUNCTIONS FROM G MATRIX- - - - - - - - - - - - - - - - - -
10005 MAT H=ZER
10010 NO=1
10020 FOR N=M1 TO 100
10030 FOR X=1 TO 27
10040 H(NO,X)=G(N,X)
10050 NEXT X
10060 IF G(N,27)<=-90 GOTO 10100
10070 IF G(N,27)>=90 GOTO 10100
10080 NO=NO+1
10090 NEXT N
10100 NO=NO+1
10101 IF M=1 GOTO 10210
10105 FOR N=M2 TO 100
10110 FOR X=1 TO 27
10120 H(NO,X)=G(N,X)
10130 NEXT X
10140 IF G(N,27)<=-90 GOTO 10180
10150 IF G(N,27)>= 90 GOTO 10180
10160 NO=NO+1
```

```
10170 NEXT N
10180 NO=NO+1
10210 H1=NO
10212 REM
10215 RETURN
10500 REM FUNCTION TRANSFER--FUNCTION MODIFICATION- - - - - - - - - - - - - -
10505 IF M=1 GOTO 10510
10507 H1=N
10510 MAT T=ZER
10520 MAT T=G
10530 FOR A=1 TO H1-1
10540 FOR X=1 TO 27
10550 G(A1,X)=H(A,X)
10560 NEXT X
10570 A1=A1+1
10580 NEXT A
10590 FOR A=A1 TO 100
10600 FOR X=1 TO 27
10610 G(A,X)=0
10620 NEXT X
10630 NEXT A
10640 IF M=2 GOTO 10670
10650 A2=L(NO,4)
10660 GOTO 10680
10670 IF L(NO+1,3)=1 GOTO 10750
10675 A2=L(NO+1,3)
10680 FOR A=A1 TO 100
10690 FOR X=1 TO 27
10700 G(A,X)=T(A2,X)
10710 NEXT X
10720 IF A2=100 GOTO 10750
10730 A2=A2+1
10740 NEXT A
10750 IF M=2 GOTO 10790
10760 D=L(NO,3)+H1-1-L(NO,4)
10770 L(NO,4)=L(NO,4)+D
10780 GOTO 10800
10790 D=L(NO,4)+H1-1-L(NO+1,3)
10800 FOR X=NO+1 TO 26
10810 IF L(X,1)=1 GOTO 10850
10820 L(X,3)=L(X,3)+D
10830 L(X,4)=L(X,4)+D
10840 NEXT X
10850 RETURN
10860 CLOSE #1
10870 END
```