NASA CONTRACTOR REPORT

NASA CR-161294

STANDARD TRANSISTOR ARRAY (STAR) - Volume 2: TEST PATTERN GENERATION

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

FINAL REPORT

N81-32389

September 14, 1979

Prepared for

NASA - George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Logic circuit testing as viewed in this report is the process of exercising 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 normal functions are produced by failures called faults introduced during the manufacturing process or that occur at random 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 emerge from an assembly 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

---

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

present in 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. Detection and location of faults is the objective of fault diagnosis testing.

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

PROBLEM STATEMENT

An automated method is desired for generating fault detection test sequences for logic circuits given a gate-level description of the circuit. Complexity of the circuits may approach five hundred or more elements including NAND gates, NOR gates, and flip-flops. Test sequences that excite race conditions in the circuit under test should not be generated. The test sequences should cover all detectable single stuck-type faults in the circuit. It is desired for the method to be user-oriented and to be implementable on small to medium scale computer systems.

SOLUTION APPROACH

A logic simulator based approach is used to solve the problem stated above. The approach is diagrammed in Figure 1. Each step of the procedure is discussed in more detail below.

Primary Test Pattern Generation

This step in TPG produces a test sequence that will detect a large percentage of the faults represented by the stuck-at fault model. This process does not require that faults be separately identified during generation of tests and is fast in terms of computer time per fault detected. The SIMLOG/TESTGN system can be used in this mode of operation. Random methods for generating test sequences may also prove useful for this purpose but will not be covered herein.

Test Sequence Evaluation

The second step of the process is 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 simulation accuracy is improved, the cost of simulation is increased. A time-based event-driven simulator provides the best accuracy, but a unit delay simulator may be useful for some circuits. Parallel or deductive fault simulators are needed for fault simulation of practical sized circuits.

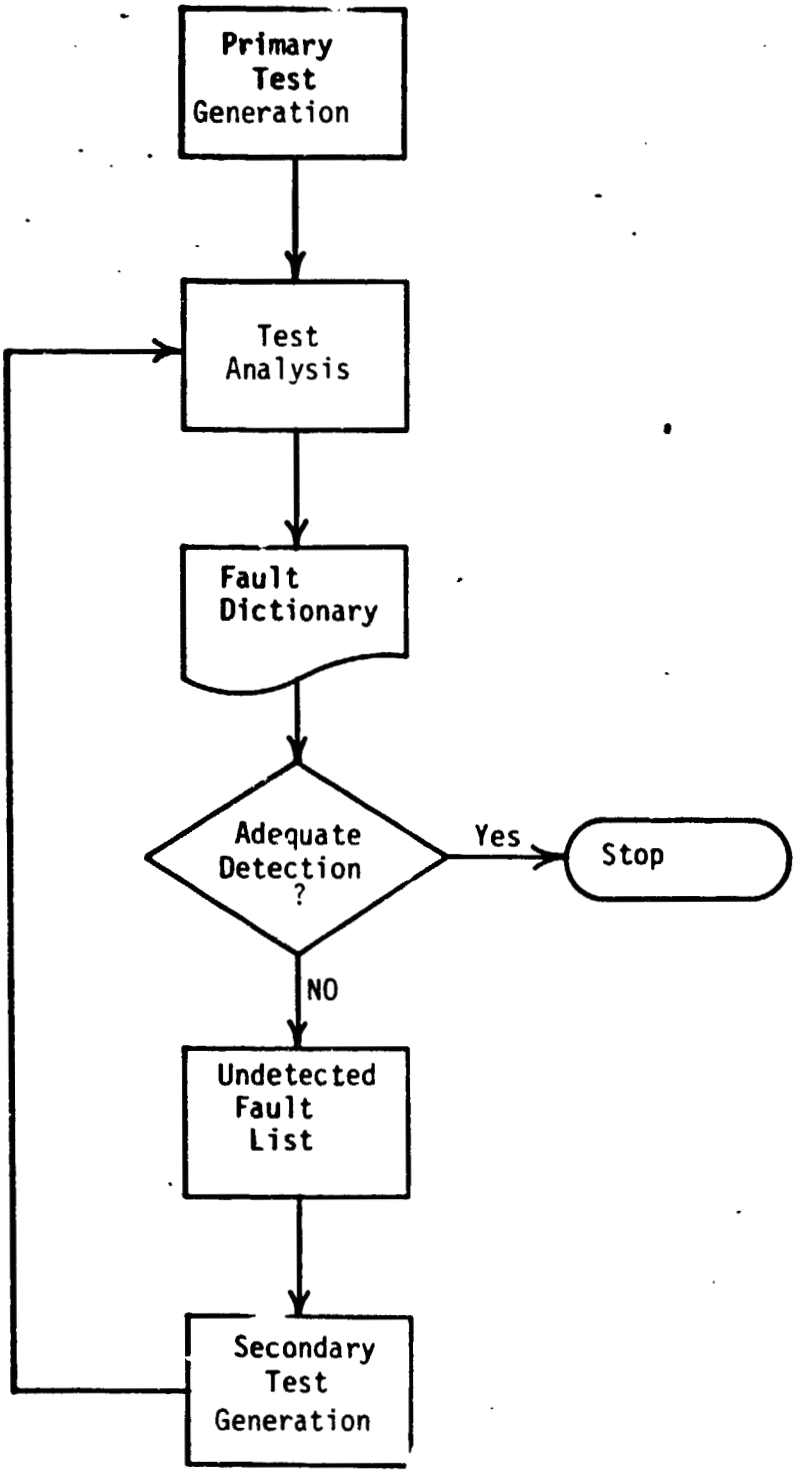


Figure 1. TEST PATTERN GENERATION SYSTEM FLOWCHART

Secondary Test Pattern Generation

The function of this step of the TPG process is to determine test sequences for those faults that are not detectable by the sequence produced during primary test pattern generation. Faults for which tests are desired are specified to the generator individually. The SIMLOG/TESTGN system can also be used here.

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.

2. LOGIC SIMULATION

This section is devoted to a description of the simulation model used in the SIMLOG logic simulator. The model has been adapted from the approach presented by Chappell [6]. However, major changes have been made in the race detection and prevention procedure; and additional logic elements have been provided in the library.

Topics covered in this section include the logic model, timing model, race analysis procedure, and oscillation detection. Fault simulation will be covered in section 3. Details on SIMLOG can be found in [7] and [8].

LOGIC MODEL

A three-valued logic model is used in the simulation procedure and will now be described. Consider the NAND gate shown in Figure 2. Each input and output of the gate is represented by an ordered pair of binary Boolean variables. The meaning of each possible combination of a variable pair is given in Table 1. A NAND gate truth table in terms of variable pairs is presented in Table 2. From the truth table, the following pair of Boolean equations can be obtained to represent a NAND gate.

$$C = A- + B-$$

$$C- = A\ B$$

Figure 2. TWO-INPUT NAND GATE

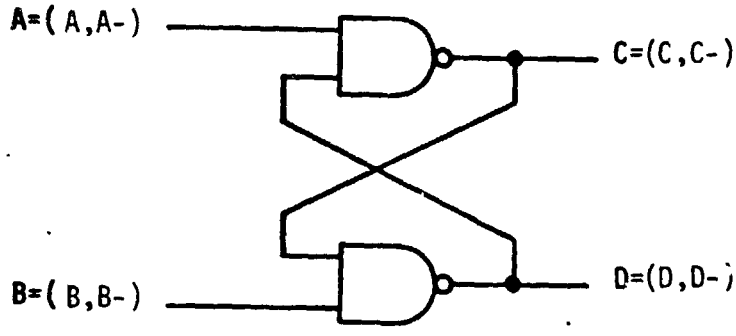


Figure 3. CROSS-COUPLED NAND GATE PAIR

TABLE 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

NAND Gate Truth Table

(A,A-)	(B,B-)	(C,C-)
(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 characteristic of the three-valued model is the ability to describe an unknown logical value as well as the usual logical 1 and logical 0 values. However, the individual 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 3.

$$C = A- + D-$$

$$C- = A\ D$$

$$D = B- + C-$$

$$D- = B\ C$$

Further description of logic elements and circuits will be deferred until after the following discussion of the timing model.

TIMING MODEL

A unit-delay timing model is used in conjunction with the above logic model. 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 3.

$$C(t,r) = A\text{-}(t) + D\text{-}(t, r\text{-}1)$$

$$C\text{-}(t,r) = A(t)\, D(t, r\text{-}1)$$

$$D(t,r) = B\text{-}(t) + C\text{-}(t, r\text{-}1)$$

$$D\text{-}(t,r) = B(t)\, C(t, r\text{-}1)$$

The above equations can be used to develop a second set of equations that describe the circuit outputs in terms of circuit inputs only for values of t ranging from 1 to τ. The value of τ is referred to as the input-time limit.

Table 3 shows the development of such an equation set for the cross-coupled NAND gates of Figure 3. In this example, $\tau = 2$ was chosen. Also, the initial states of C and D are assumed unknown, i.e., (0,0).

Note in Table 3 that the value of r is incremented only for each new set of equations that result. On the other hand, t is incremented only after a stable set of equations has been reached. These procedures follow from the unit-delay assumption and from an assumption that input changes occur only while the circuit is in a stable state. The latter assumption will be followed throughout the remainder of this work.

The above assumptions imply that the increment of ripple-time for a given input-time t represents the time required for the circuit to

TABLE 3

Equation Development for Cross-Coupled NAND Gate Pair

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

reach stability following an input change at time t. Furthermore, it represents the minimum amount of time that must occur between t and t + 1.

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

The value of r and the parenthesis will be suppressed when writing stable equations as illustrated below.

C1 = A-1

C-1 = A1 B-1

D1 = B-1

D-1 = A-1 B1

Special steps may be necessary in order that stability be reached in sequential circuits. As can be seen in Table 3 the equation set starts repeating at r = 6. 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 excites 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 a later subsection.

LOGIC ELEMENT DESCRIPTIONS

The SIMLOG logic simulator provides NAND gates, NOR gates, unit-delay devices, and edge-triggered D flip-flops as standard logic elements. A description of each of these elements in terms of the above models is given below.

NAND Gates

NAND gates may have two or more inputs. Inventers are realized by applying a common input to both inputs of a two-input NAND (or NOR). The equation pair below represents the unstable or transient behavior of the three-input NAND shown in Figure 4. Generalization to an n-input NAND is straightforward.

$$D(t,r) = A\text{-}(t,r\text{-}1) + B\text{-}(t,r\text{-}1) + C\text{-}(t,r\text{-}1)$$

$$D\text{-}(t,r) = A(t,r\text{-}1)\ B(t,r\text{-}1)\ C(t,r\text{-}1)$$

The stable or steady-state description of the device is the following.

$$Dt = A\text{-}t + B\text{-}t + C\text{-}t$$

$$C\text{-}t = AtBtCt$$

NOR Gate

NOR gates may have two or more inputs. The equation pair below represents the unstable or transient behavior of the three-input NOR shown in Figure 5. Again generalization to the n-input case is obvious.

$$D(t,r) = A\text{-}(t,r\text{-}1)\ B\text{-}(t,r\text{-}1)\ C\text{-}(t,r\text{-}1)$$

$$D\text{-}(t,r) = A(t,r\text{-}1) + B(t,r\text{-}1) + C(t,r\text{-}1)$$

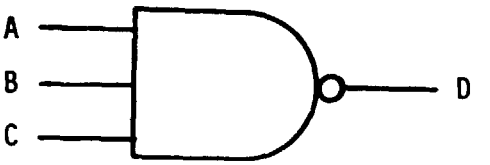


Figure 4. THREE-INPUT NAND GATE

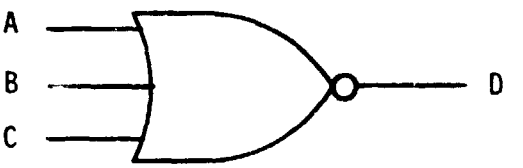


Figure 5. THREE-INPUT NOR GATE

The stable or steady-state behavior is as follows.

$$Dt = A-t + B-t + C-t$$

$$D-t = AtBtCt$$

Unit-Delay Device

A unit-delay device is a one-input, one-output device as illustrated in Figure 6. The following equation pair describes its behavior in the unstable or transient case.

$$B(t,r) = A(t,r-1)$$

$$B-(t,r) = A-(t,r-1)$$

The stable or steady-state model becomes

$$Bt = At$$

$$B-t = A-t$$

Edge-Triggered (Negative) D Flip-Flop

The negative edge-triggered D flip-flop is a two-input, two-output device as shown in Figure 7. Data to be latched or stored is applied to input D. A clock or other triggering signal is applied to input C. Triggering occurs on a logic one to logic zero (negative) transition. Output Q provides the uncomplemented version of the latched value of D, whereas output QBAR provides the complement of D.

Transient behavior of the flip-flop is given by the equation pair below for output Q. The right hand sides are reversed for QBAR.

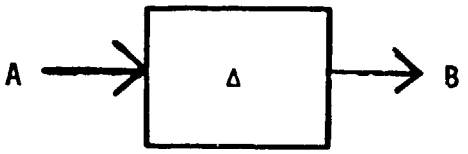


Figure 6. UNIT-DELAY DEVICE

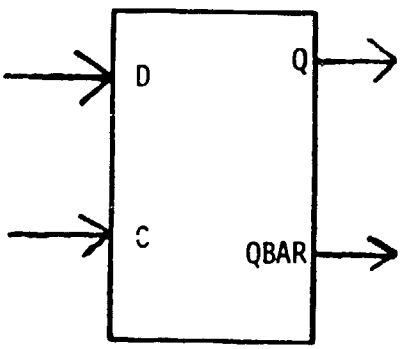


Figure 7. NEGATIVE EDGE-TRIGGERED D FLIP-FLOP

$$Q(t,r) = D(t-1,r-1)C^{-}(t)C(t-1)$$
$$+ Q(t-1)[C^{-}(t)C^{-}(t-1) + C(t)C^{-}(t-1) + C(t)C(t-1)]$$

$$Q^{-}(t,r) = D^{-}(t-1,r-1)C^{-}(t)C(t-1)$$
$$+ Q^{-}(t-1)[C^{-}(t)C^{-}(t-1) + C(t)C^{-}(t-1) + C(t)C(t-1)]$$

The stable or steady-state equations are the same as above with r and r-1 suppressed.

RACE ANALYSIS

Race conditions exist in circuits that contain cross-coupled NAND gates or cross-coupled NOR gates. Input sequences that excite these races can be easily detected, and a procedure for accomplishing such detection is discussed below. Race conditions may also exist in other circuit configurations but detection of this type race is much more difficult and will not be explicitly considered here. However, this type race may often produce an oscillation which is the topic of a later discussion.

It is shown in [6] that a race condition is first indicated when both the equations for C- and D or D- and C change from the previous ripple-time for the circuit in Figure 3. When one of these conditions is detected, C- and D- can be systematically modified so that input changes are restricted to prevent a race condition from being excited. Race analysis consists of the detection and the prevention of race conditions.

For the circuit in Figure 3, Chappell [6] has shown that if the equation for C- 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 alone is applied to the circuit in Figure 8. Hence, the race detection procedure adopted here is to check C- and D plus D- and C for changes in all cross-coupled NAND gates in a circuit under analysis. An analogous approach is used for cross-coupled NOR gates.

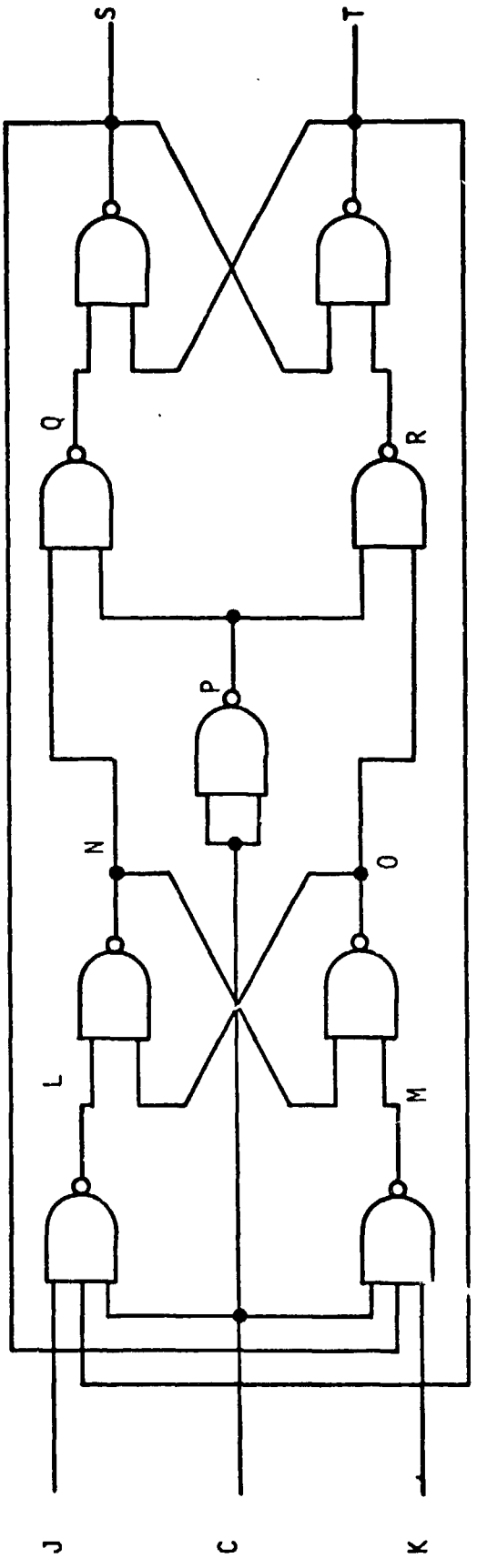


Figure 8. JK Master-Slave Flip-Flop.

When a race condition is detected in a pair of cross-coupled NAND gates, equations C- or D and D- or C are modified in such a manner to eliminate input sequences that would cuase the race to be excited. The equation modification rules adopted here are given in Table 4. Rules I.A and II.A are the same as those given in [6]. An explanation of the rules will now be given.

Consider the possible combinations of outputs of the cross-coupled NAND gates of Figure 3. The output combinations that do not indicate a race condition are shown in Figure 9a. Possible race conditions are indicated by those combinations given in Figure 9b.

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

$$C \supseteq D- \tag{1a}$$

$$D \supseteq C- \tag{1b}$$

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

$$C \subseteq D- \tag{2a}$$

$$D \subseteq C- \tag{2b}$$

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 4 modify the equations so the desired relationships are satisfied. Table 5 shows the application of these rules to the analysis of the cross-coupled NAND gate circuit.

TABLE 4

Race Prevention Rules for Cross-Coupled NAND Gates

I. Modification of C- or D.

A. If $C\text{-}(t,r) \neq 0$ and $D(t,r) \neq 0$

Then $C\text{-}(t,r) = C\text{-}(t,r) \cdot D(t,r)$.

B. If $C\text{-}(t,r) \neq 0$ and $D(t,r) = 0$,

Then $D(t,r) = C\text{-}(t,r)$.

C. If $C\text{-}(t,r) = 0$, then no change.

II. Modification of D- and C.

A. If $D\text{-}(t,r) \neq 0$ and $C(t,r) \neq 0$,

Then $D\text{-}(t,r) = D\text{-}(t,r) \cdot C(t,r)$.

B. If $D\text{-}(t,r) \neq 0$ and $C(t,r) = 0$, then $C(t,r) = D\text{-}(t,r)$.

C. If $D\text{-}(t,r) = 0$, then no change.

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

(b) Possible Race Conditions

Figure 9. Analysis of Cross-Coupled NAND Gate Outputs.

TABLE 5

Equation Development with Race Analysis for Cross-Coupled NAND Gates

t	r	C(t,r)	C-(t,r)	D(t,r)	D-(t,r)
0	0	0	0	0	0
1	1	A-(1)	0	B-(1)	0
	2	A-(1)	A(1)B-(1)	B-(1)	A-(1)B(1)
	∞	A-(1)	A(1)B-(1)	B-(1)	A-(1)B(1)
2	3	A-(2)+A-(1)B(1)	A(2)B-(1)	B-(2)+A(1)B-(1)	A-(1)B(2)
	3R		A(2)B-(2)B-(1)+ A(2)A(1)B-(1)		A-(2)A-(1)B(2)+A-(1)B(2)B(1)
	4	A-(2)+A-(1)B(2)B(1)	A(2)B-(2)+A(1)B-(1)	B-(2)+A(2)A(1)B-(1)	A-(2)B(2)+A-(1)B(2)B(1)
	4R		A(2)B-(2)+A(1)B-(1)		A-(2)B(2)+A-(1)B(2)B(1)
	∞	A-(2)+A-(1)B-(2)B(1)	A(2)B-(2)+A(1)B-(1)	B-(2)+A(2)A(1)B-(1)	A-(2)B(2)+A-(1)B(2)B(1)

Now consider the interpretation of the equations in Table 5 for t=2, r=4. These equations represent all race-free ways of controlling the circuit outputs in two input-time steps. In particular, the equation

$$C(2,6) = A\text{-}(2) + A\text{-}(1)B(2)B(1)$$

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.

Race analysis for cross-coupled NOR gates proceeds in a similar manner. Table 6 shows the race prevention rules for cross-coupled NOR gates as shown in Figure 10.

OSCILLATION ANALYSIS

Equation oscillation may occur during the circuit simulation procedure even if race conditions are prevented in cross-coupled NAND or NOR gates. This type 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 such oscillation conditions has been developed. However, such oscillations can be handled by setting an upper limit on the number of ripple-time increments allowed for each input-time step. An oscillation is assumed to exist if the equation sets do not stabilize before the ripple-time increment exceeds the established limit.

TABLE 6

Race Prevention Rules for Cross-Coupled NOR Gate

I. Modification of C or D-.

A. If $C(t,r) \neq 0$ and $D\text{-}(t,r) \neq 0$,

then $C(t,r) = C(t,r)\ D\text{-}(t,r)$.

B. If $C(t,r) \neq 0$ and $D\text{-}(t,r) = 0$,

then $D\text{-}(t,r) = C(t,r)$.

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

II. Modification of D or C-.

A. If $D(t,r) \neq 0$ and $C\text{-}(t,r) \neq 0$,

then $D(t,r) = D(t,r)\ C\text{-}(t,r)$.

B. If $D(t,r) \neq 0$ and $C\text{-}(t,r) = 0$,

then $C\text{-}(t,r) = D(t,r)$.

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

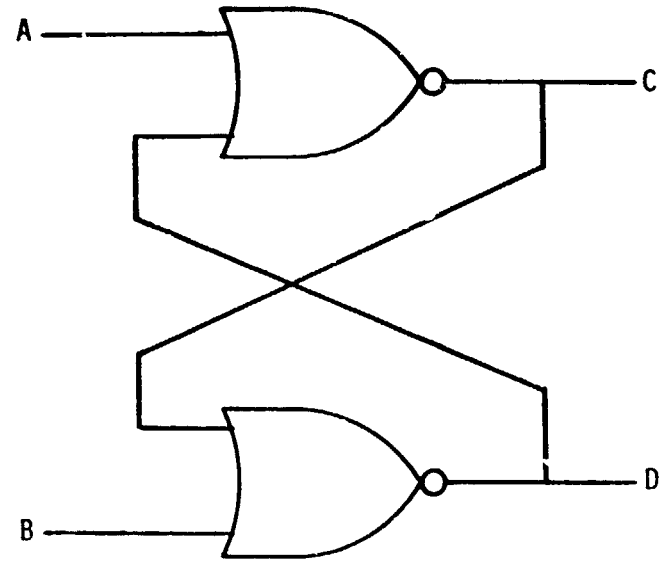


Figure 10. CROSS-COUPLED NOR GATE PAIR

SIMULATION PROCEDURE

Given the models and analysis procedures described above, a logic circuit simulation procedure can be described. An overview of the procedure used in SIMLOG will now be presented. The overview will be restricted to NAND gate logic element for simplicity of presentation.

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_1, I_i^{ℓ}) and (J_j, J_j^{ℓ}) 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.

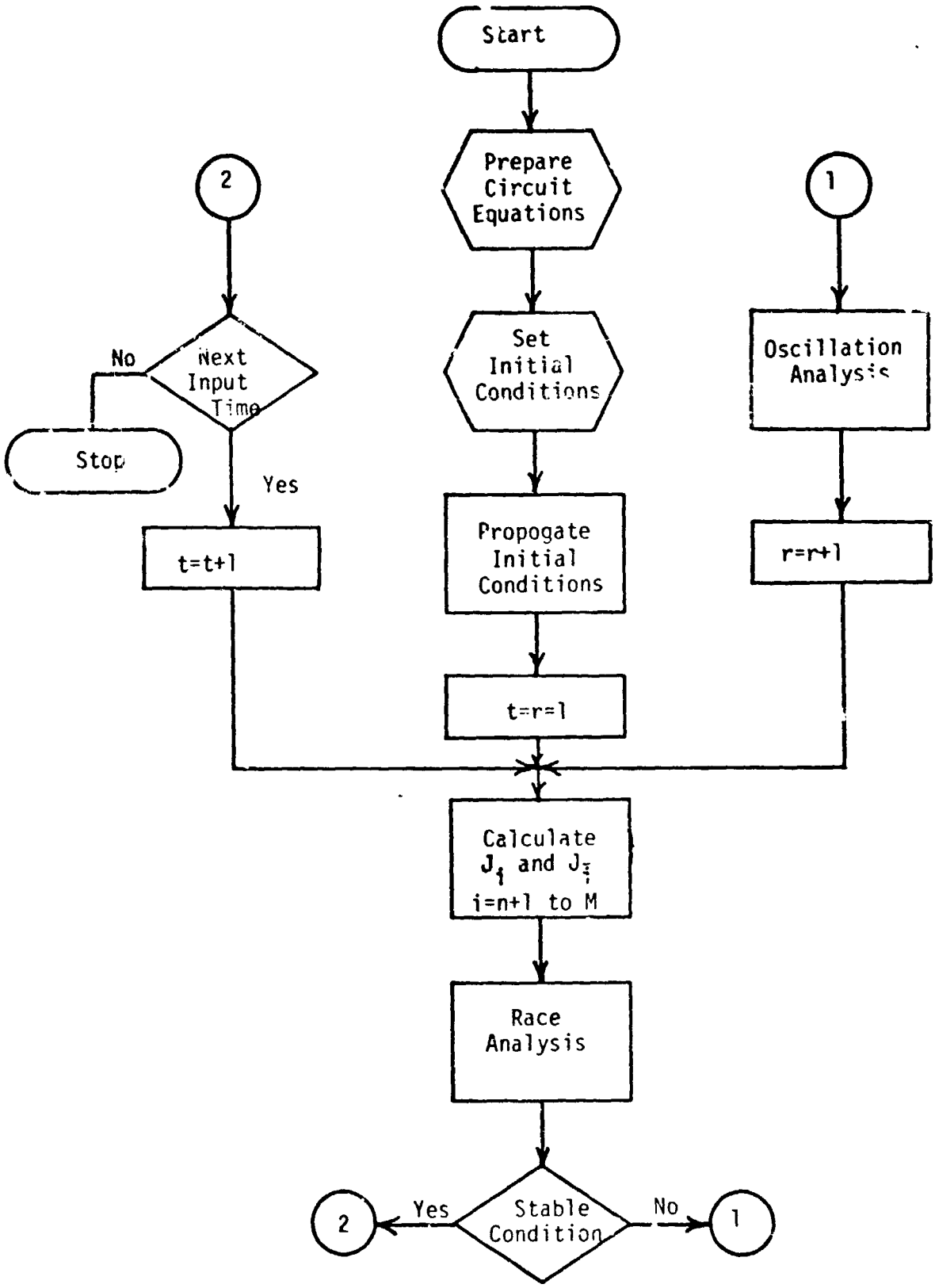
$$J_j(t,r) = \sum_{k \epsilon K_j} I_{\overline{k}}(t) + \sum_{\ell \epsilon L_j} J_{\overline{\ell}}(t, r-1) \qquad (3a)$$

$$J_{\overline{j}}(t,r) = \prod_{k \epsilon K_j} I_k(t) \cdot \prod_{\ell \epsilon L_j} J_{\ell}(t, r-1) \qquad (3b)$$

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

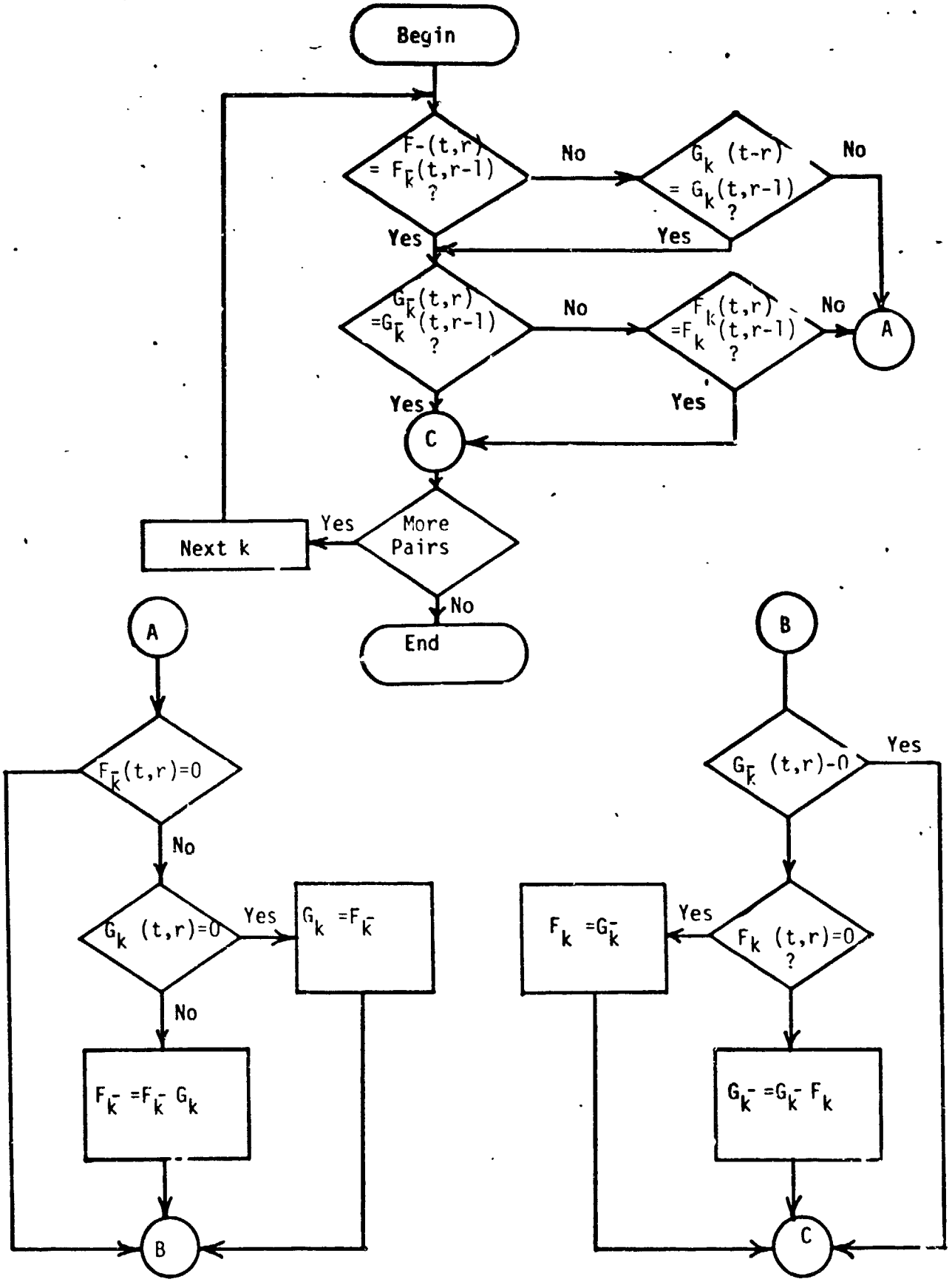
and L_j = {Gate outputs that are inputs of gate j}

The circuit analysis procedure illustrated previously is flowcharted in Figure 11 in terms of the notation used in (3). Also, cross-coupled NAND gate outputs are denoted F and G in the race analysis procedure flowchart.



(a) Main Procedure

Figure 11. Simulation Procedure Flowchart



(b) Race Analysis Procedure

Figure 11. (Cont'd)

3. STARTING STATE SPECIFICATION

A major advantage of the logic model introduced in Section 2 is that a representation exists for describing a logic signal in an unknown state. This allows simulation of sequential circuits from an unknown starting state. However, sequential circuit examples have been found that cause difficulties when applying the simulation procedure of Section 2 if the starting states are unknown. This Section will be devoted to a presentation of some of these difficulties and to a discussion of how the problems are handled in SIMLOG.

PROBLEM CIRCUITS

The circuit shown in Figure 12 is from [6] and has only one stable starting state. Establishment of this state must be accomplished manually before the simulation procedure of Section 2 can be applied. Hence, the unknown starting state will not yield meaningful results.

A 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 8. Simulation 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. When a specific starting state is assumed for the JK flip-flop, the analysis procedure yields the proper results.

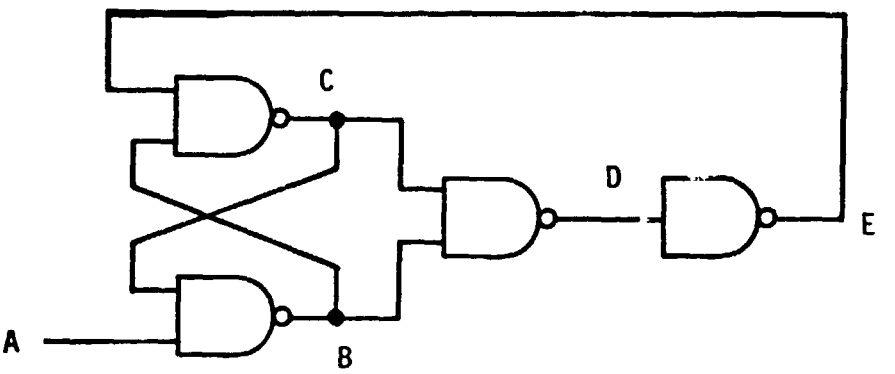


Figure 12. Self-initializing Circuit [6].

The simulation of large sequential circuits cannot be accomplished effectively unless the simulation procedure can start with the circuit in an unspecified state. Several starting state options are available in SIMLOG to handle this problem. These options are described below with the exception of the unknown case which is self explanatory.

CROSS-COUPLED Gate Variables

It has been found that by assigning symbols other than 0 or 1 to represent the starting state of selected nets meaningful simulation results can be obtained. These symbols can be replaced after simulation is complete by the desired starting state representation. Nets selected for the assignment of such symbols are those corresponding to the output lines of cross-coupled gates. Other nets are assigned as unknown.

Two assignment modes of this type are available. One mode assigns unique symbols to each net in a pair whereas the other mode assigns the same symbol to both nets. For example, consider the cross-coupled NAND gates of Figure 3. In the DOUBLE CROSS-COUPLED VARIABLES mode, the starting state of net C is specified as (CØ,C-Ø) and the starting state of net D as (DØ,D-Ø). In the SINGLE CROSS-COUPLED VARIABLE mode, the assignments would be (CØ,C-Ø) and (C-Ø,CØ) for C and D, respectively.

An extension of this concept is also available which assigns symbols as the starting state of each net (inputs excluded) in a circuit. For this case, net X is assigned (XØ,X-Ø) as its starting state.

OTHER POSSIBLE ASSIGNMENTS

A mode is also provided which allows the SIMLOG user to specific constants (0 or 1) as the starting state of user selected nets including circuit input nets. Nets not explicitly assigned constant values are implicitly assigned as unknown.

Another mode permits the user to assign Boolean equations in sum of products form as the starting state of user selected nets. This mode provides a generalized starting state assignment capability.

4. FAULT SIMULATION

The logic simulation procedure described in the previous section can be modified to simulate stuck-type faults. This section contains descriptions of the model for stuck-at faults and of the corresponding fault insertion procedures used in the SIMLOG program.

FAULT MODEL

Stuck-type faults can be modeled in the simulation procedure described previously by an ordered pair of Boolean variables. Consider for example the two-input NAND gate of Figure 2. A stuck-at-0 fault on input A is represented by the ordered pair (A*,A-*) where (A*,A-*)= (1,0) means the fault is present and (A*,A-*)=(0,1) means the fault is not present. The (0,0) and (1,1) combinations are not assigned meanings in this context.

A stuck-at-1 fault on input B of the NAND gate is represented by the ordered pair (B!,B-!). The (1,0), (0,1), (0,0) and (1,1) are assigned similar meanings to those given above for stuck-at-0 faults.

FAULT INSERTION

Faults are inserted in a logic circuit by modifying the appropriate equations corresponding to the element to be faulted. Again, consider the two-input NAND gate of Figure 2. Table 7 shows the fault-free equation pair, the equation pairs for stuck-at-0 and stuck-at-1 faults

TABLE 7

Fault Insertion in Two-Input NAND Gates

Fault Free	A Stuck-at-0	A Stuck-at-1	C Stuck-at-0	C Stuck-at-1
C = A- + B	C = (A- + A*) + B-	C = (A- A-!) + B-	C = (A- + B-)C-*	C = (A- + B-) + C!
C-= A B	C-= (AA-*)B	C-= (A + A!)B	C-= (AB) + C*	C-= (AB)C-!

on input A, and the equation pairs for stuck-at-0 and stuck-at-1 faults on the output C.

An examination of the equations in Table 7 reveals that fault insertions can be accomplished by ANDing or ORing the appropriate fault-free variable or expression. For example, a stuck-at-0 fault on input A is inserted by first ORing fault variable A* with fault free variable or expression A- to produce (A- + A*). Next, the fault variable A-* is ANDed with fault free variable or expression A to produce (AA-*). Finally, the gate output equations are computed in the usual way except that (A- + A*) is used in place of A- and (AA-*) is used in place of A.

The equations corresponding to faulted outputs are similarly computed. For example, a stuck-at-0 fault on the output C is inserted by first computing the fault-free output expressions (A- + B-) and (AB). Fault insertion is completed by computing (A- + B-)C-* and (AB) + C*.

Fault insertion for NOR gates is accomplished similarly and is detailed in Table 8. Fault insertion for unit-delay elements and D flip-flops is shown in Tables 9 and 10, respectively.

Further illustrations of fault simulation is given by considering the cross-coupled NAND gates shown in Figure 3. The following set of equations describe the circuit with a stuck-at-1 fault on input B.

TABLE 8

Fault Insertion in Two-Input NOR Gates

Fault Free	A Stuck-at-0	A Stuck-at-1	C Stuck-at-0	C Stuck-at-1
C = A- B-	C = (A- + A*)B-	C = (A- A-!) B-	C = (A- B-) C-*	C = (A- B-) + C!
C-= A + B	C-= (AA-*) + B	C-= (A + A!) + B	C-= (A + B) + C*	C-=)A + B) C-!

TABLE 9

Fault Insertion in Unit-Delay Element

Fault-Free	A Stuck-at-0	A Stuck-at-1	B Stuck-at-0	B Stuck-at-1
B = A	B = AA-*	B = A + A!	B = AB-*	B = A + B!
B-= A-	B-= A- + A*	B-= A- A-!	B-= A- + B*	B-= A- ∂-!

TABLE 10

Fault Insertion In D Flip-Flop

Fault Condition	Equation Pair†
Fault Free	Q(t,r) = D(t-1,r-1)X + Q(t-1)Y Q-(t,r) = D-(t-1,r-1)X + Q-(t-1)Y
D Stuck-at-0	Q(t,r) = D(t-1,r-1)X D-* + Q(t-1)Y Q-(t,r) = D-(t-1,r-1)X + D* + Q(t-1)Y
D Stuck-at-1	Q(t,r) = D(t-1,r-1)X + D! + Q(t-1)Y Q-(t,r) = D-(t-1,r-1)X D-! + Q-(t-1)Y
C Stuck-at-0	Q(t,r) = D(t-1,r-1)X C-* + Q(t-1)[Y+C*] Q-(t,r) = D-(t-1,r-1)X C-* +Q-(t-1)[Y+C*]
C Stuck-at-1	Q(t,r) = D(t-1,r-1)X C-! + Q(t-1)[Y+C!] Q-(t,r) = D-(t-1,r-1)X C-! + Q-(t-1)[Y+C!]
Q Stuck-at-0	Q(t,r) = D(t-1,r-1)X Q-* + Q(t-1)Y Q-(t,r) = D-(t-1,r-1)X + Q* + Q-(t-1)Y
Q Stuck-at-1	Q(t,r) = D(t-1,r-1)X + Q! + Q(t-1)Y Q-(t,r) = D-(t-1,r-1)X Q-! + Q-(t-1)Y

† X = C-(t)C(t-1)

Y = C(t)C(t-1) + C(t)C-(t-1) + C-(t)C-(t-1)

$$C\ (t,r) = A\text{-}(t) + D\text{-}(t,\ r\text{-}1)$$

$$C\text{-}(t,r) = A\ (t)D\ (t,r\ -1)$$

$$D\ (t,r) = B\text{-}!\ B\text{-}(t) + C\text{-}(t,\ r\text{-}1)$$

$$D\text{-}(t,r) = B!C(t,\ r\text{-}1) + B\ (t)C(t,r\ -1)$$

Propagation of the above equations is shown in Table 11 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.

TABLE 11

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-(1)	0	B-! B-(1)	0
	2	A-(1)	A(1)B-! B-(1)	B-! B-(1)	B!A-(1)+B(1)A-(1)
		A-(1)	A(1)B-! $B^0(1)$	B-! B-(1)	B!A-(1)+B(1)A-(1)
2	3	A-(2)+B!A-(1)+B(1)A-(1)	A(2)B-! B-(1)	B-!B-(2)+A(1)B-!B-(1)	B!A-(1)+B(2)A-(1)
	3R		A(2)B-!B-(1)B-(2) + A(2)A(1)B-!B-(1)		B!A-(1)+A-(2)B(2)A-(1) + B(2)B(1)A-(1)
	4	A-(2)+B(2)B(1)A-(1) + B!A-(1)	A(2)B-!B-(2) + A(2)A(1)B-!B-(1)	A(2)A(1)B-!-(1) + B-!B-(2)	B!A-(2)+B!A-(1)+A-(2)B(2) + B(1)B(2)A-(1)
	4R		A(2)B-!B-(2) + A(2)A(1)B-!B-(1)		B!A-(2)+A-(2)B(2)+B!A-(1) + B(1)B(2)A-(1)
	5	A-(2)+B!A-(1) + B(1)B(2)A-(1)	A(2)B-(2)B_1 + A(2)A(1)B-(1)B-!	A(2)A(1)B-(1)B-! + B-(2)B-!	B!(A)-(2)+A-(1)+A-(2)B(2) + B(2)B(1)A-(1)

5. TEST PATTERN GENERATION

Test sequences for a circuit can be generated from the results of a simulation procedure such as that described in the previous sections. Two methods of generation are possible. One method produces sequences for input faults and does not require the use of fault simulation. The other method produces sequences for specified faults on any single net but requires that fault simulation be used. Each of these methods will now be described. Both methods are available in TESTGN.

TESTS FOR INPUT FAULTS

An efficient method for generation test sequences is to produce test sequences for stuck-at-0 and stuck-at-1 faults on each circuit input for 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 approach to be described later.

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.

$$O_J = A + BI_j + CI_{\bar{i}}$$

$$O_j = D + EI_j + FI_{\bar{i}}$$

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

The desired test function is given below.

$$\sigma'_{I_i} = (BF + CE)\ (I_i + I_{\bar{i}}) \tag{5}$$

It should be emphasized that fault-free circuit equations are used in obtaining Equation (5).

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

t = 1:

$$\sigma_A = B\text{-}(1)\ [A(1) + A\text{-}(1)]$$
$$= A(1)B\text{-}(1) + A\text{-}(1)B\text{-}(1)$$

t = 2:

$$\sigma_A^2 = [B\text{-}(2) + A(1)B\text{-}(1)][A(2) + A\text{-}(2)]$$
$$= A(2)B\text{-}(2) + A\text{-}(2)B\text{-}(2) + A(2)\ A(1)B\text{-}(1)$$
$$+ A\text{-}(2)A(1)B\text{-}(1)$$

Selection of tests from (5) 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.

TESTS FOR SPECIFIC FAULTS

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

$$F = U + V\phi + W\phi\text{-}$$

$$F\text{-} = X + Y\phi + Z\phi\text{-}$$

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.

$$F_{\phi}^{t} = VZ + WY \qquad (6)$$

For an example, consider fault B stuck-at-1 in the cross-coupled NAND gates of Figure 3. The results will be shown for only output C even though similar results can be obtained for output D.

At t = 1:

$$C(1) = B\text{-}(1)B\text{-}!$$

$$C\text{-}(1) = A\text{-}(1)B(1) + A\text{-}(1)B!$$

$$\mathfrak{e}_{B!}^{1} = A\text{-}(1)B\text{-}(1)$$

At t = 2:

$$C(2) = B(2)B\text{-}! + A(2)A(1)B(1)B\text{-}!$$

$$C\text{-}(2) = B(2)A(2) + B(2)B(1)A(1) + A(2)B! + A(1)B!$$

Therefore:

$$\mathfrak{e}_{B!}^{2} = A\text{-}(2)B\text{-}(2) + A\text{-}(1)B\text{-}(2)$$

6. CONCLUSIONS AND RECOMMENDATIONS

Approaches to logic simulation and test pattern generation have been described that can be used with both combinational and sequential logic circuits. Simulation of fault-free circuits and of circuits with stuck-type faults can be accomplished. 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.

Future work is recommended on generalization of the timing model and functional representation of circuit segments. The former would produce a more accurate simulation while the latter is one approach that could possibly reduce computation time and memory requirements of the procedure.

Further work on the starting state problem and 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.

It is also recommended that the fault model be generalized. This would impact both the simulation procedure and the test generation procedure.

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 1073, 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.

7. B. D. Carroll, "SIMLOG/TESTGN User's Guide," Final Report-Vol. 2-Addendum 1, Contract NAS8-31572, Electrical Engineering Department, Auburn University, Auburn, Alabama, September 14, 1979.

8. B. D. Carroll, "SIMLOG/TESTGN Programmer's Guide," Final Report-Vol. 2-Addendum 2, Contract NAS8-31572, Electrical Engineering Department, Auburn University, Auburn, Alabama, September 14, 1979.