A MATRIX MODEL OF DIGITAL SYSTEMS AND ITS APPLICATION TO AUTOMATIC TEST GENERATION

By

CHARLOTTE COUCH ACKEN

Bachelor of Science
University of Arkansas
Fayetteville, Arkansas
1969

Master of Science
University of Arkansas
Fayetteville, Arkansas
1971

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1977

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
May, 1982

A MATRIX MODEL OF DIGITAL SYSTEMS AND ITS
APPLICATION TO AUTOMATIC
TEST GENERATION

Thesis Approved:

Ronald P. Rhoten
Thesis Adviser

James W. Maxwell

Marvin E. Daniel

Robert J. Mulholland

C.M. Bacon

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

My sincere appreciation is extended to Dr. Ronald P. Rhoten, chairman of my doctoral committee and my thesis adviser, for his invaluable assistance throughout the course of my graduate education in electrical engineering. I am grateful to the other members of my doctoral committee, Dr. C. M. Bacon, Dr. M. E. Daniel, Dr. J. W. Maxwell, and Dr. R. J. Mulholland, for their critical reviews of the manuscript and for their encouragement and friendship.

Financial support for this research has been provided by Sandia National Laboratories. Sandia management directly involved in providing me the opportunity to conduct this work were D. M. Olson, D. M. Schuster, C. W. Gwyn, A. A. Lieber, M. E. Daniel, and W. F. Roherty. Their exhibited confidence in my abilities are much appreciated.

Special thanks are extended to the staff of the Sandia National Laboratories technical library, word processing center, and graphic arts division for their assistance. The typing of the final copy was performed by Charlene Fries, to whom I am most grateful.

Well deserved gratitude is extended to my parents, whose personal sacrifice, encouragement, and understanding have inspired me throughout my education. I am thankful for my wonderful husband, John, who has provided loving encouragement and enthusiastic confidence in me during the preparation of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION AND BACKGROUND

Physical faults can occur at any point during the life of a digital system. Generally speaking, a fault can be defined as anything which causes the system to operate in a manner other than that for which it was designed. Faults can originate during the manufacture, assembly, storage, and/or service phase of the system life. Open circuits, short circuits, and incorrect impedance are example faults. Those which alter the magnitude of a circuit parameter with a resultant modification in circuit speed, current, or voltage are termed parametric faults. Faults which alter the logical behavior of the system are termed logical faults. Because faults can, and do, occur at any time during the life of a digital system, it is necessary to determine, via some testing procedure, the operational status of the device.

A test for a logical fault is defined to be a set of inputs (or input sequences) for which a faulty system will exhibit an unintended logical behavior. Only tests for logical faults will be considered in this research.

The advent of large-scale integration (LSI) and very large-scale integration (VLSI), as well as the increased complexity of digital integrated circuits (ICs), has made manual test generation extremely time-consuming, unreliable, and physically difficult. For this reason, automatic test generation (ATG) algorithms are desirable. However, most ATG algorithms are also constrained by the size and complexity of present day ICs. An obvious algorithm for ATG for combinational circuits is to generate all possible input combinations and compare the output of the device under test (DUT) with a known output. This procedure detects all possible logical faults within the system. The disadvantage to this approach becomes apparent as the number of primary inputs, n, increases. Assuming a testing system with a stimulus/response cycle of 1 microsecond, it would take 1 second to exhaustively test a combinational logic circuit with 20 primary inputs. The amount of time required rises exponentially as the number of primary inputs increases, and for n = 50, the time required becomes 35 years!

One approach to narrowing the problem to feasible limits is to consider a subset of all possible faults whose detection will result in a high confidence that all likely faults of interest have been detected. The fault subset is generally defined via a logical fault model.

There are three basic requirements of any logical fault model. The model should indicate what effect the fault will impose upon the operation of the circuit, how many such faults are assumed to be present at one time, and how long the fault is present. The most common logical fault model (also to be utilized in this research) is the single permanent stuck-at model. This model assumes that only one primary input/output or internal signal line in the circuit will be permanently stuck at logical zero or logical one during the duration of any test.

If a circuit has n primary input lines and p internal signal lines (including primary outputs), then the total possible number of single faults (i.e., occurring one at a time) is 2 (n + p) since each line could be stuck-at-zero or stuck-at-one.

If two or more of the signal lines are simultaneously faulty, there are a total of $3^{n+p} - 1$ such multiple faults to consider. Because this number tends to become large as circuits increase in size, this multiple fault model becomes intractable. Even though the multiple fault model is a valid model, it has not been utilized in practical implementations because of the exponential growth in the number of faults that have to be considered.

One method of generating tests for logical faults is to have a test engineer determine input vectors which would cause a change in signal value on every signal line in the device under test. To achieve this, as well as try to minimize the test length and perform fault location simultaneously, would require a lot of detailed and repetitive work--the type of work that is more efficiently done by a computer.

Automatic test generation refers to the process of generating a set of inputs to an integrated circuit via a computer algorithm. Most algorithms utilize information about the system topology and/or behavior and the faults for which tests are being generated. Almost all automatic test generation methods are based upon models of the digital circuit in addition to fault models, some of which were discussed earlier.

Many digital system models exist. The two utilized most extensively for automatic test generation are the gate level logic diagram and the register-transfer level (RTL) diagram. The gate level logic diagram characterizes the system as an interconnection of one or more of the basic logic gates (AND, OR, NOT, NAND, NOR). An RTL diagram characterizes the circuit as an interconnection of functional blocks such as registers, counters, or ALUs. Functional level test generation utilizes an RTL circuit model and attempts to generate tests which will detect whether or not the functional blocks are performing the function for which they were designed (shifting, counting, ect.). Modeling the circuit at the register-transfer level tends to decrease the execution time and memory requirements for ATG, compared to that required for ATG algorithms which utilize a gate level circuit description. However, functional level test generation methods lack the ability to isolate faults to a gate or signal level, may generate invalid tests, and suffer from a lack of adequate fault models.

The efficiency of ATG algorithms which utilize a gate level circuit description depends upon how much information concerning the circuit behavior and topology is passed and how concisely the information is formulated for use in the algorithm. Completely random ATG formulates no information about the circuit behavior and topology to guide it in generating tests. Other algorithms formulate the behavior and/or topological information in various ways.

Chapter II is a review of some of the best known and most widely used gate level automatic test generation methods. The manner in which the behavior and/or topological information is formulated and utilized by each is reviewed, and the advantages and weaknesses of each method and its underlying mathematical formulation are pointed out.

A matrix formulation for digital integrated circuits is introduced in Chapter III. This representation contains the information about the logical behavior and topology of a circuit necessary for gate level automatic test generation. The concise matrix formulation can be easily stored in and manipulated by a computer. Characteristics of and extensions to the formulation are given in Chapter IV.

Chapter V illustrates several ways in which the matrix representation can be utilized to automatically generate tests. Approaches using path sensitization and Boolean equation manipulation for test generation are presented.

Along with the thesis conclusions, Chapter VI contains suggestions for other possible uses of the matrix representation of a digital system. These include the analytical detection of circuit redundancy and the use of the matrix model as a structure in which to perform testability analysis.

Numerous examples are included in Chapters III through VI to illustrate concepts and techniques.

CHAPTER II

REVIEW OF RELATED LITERATURE

Introduction

The first ATG algorithms were based upon gate level circuit models and the single permanent stuck-at fault model. Boolean equation manipulation was an early ATG technique followed by path sensitization approaches. Most of today's gate level ATG algorithms are based upon one or a combination of both these procedures.

In order to fully understand why certain characteristics of digital system models such as topology, behavior, and compact representation are essential to ATG, a close look at digital circuit test generation methods is necessary. Two path sensitization approaches are reviewed, the D-algorithm (Roth, 1966) and critical path sensitization (Thomas, 1971). One equation manipulation technique, the Boolean difference approach (Sellers, Hsiao, and Bearnson, 1968) is outlined. Emphasis will be placed on the type of circuit information needed for the technique and the way the information is formulated.

Path Sensitization

The D-algorithm (Roth, 1966) has formed the basis for most of the ATG programs in use today because it introduced the fundamental concept of path sensitization. Variants of the original path sensitization procedure have proven to be successful.

As it was originally conceived, path sensitization was to be performed on a gate level model of the circuit and generate tests for stuck-type faults. Others have used the idea of path sensitization on partitioned blocks of logic circuitry and on higher level primitives such as registers, counters, etc. (Breuer and Friedman, 1980; Bennetts, Brittle, Prior, and Washington, 1975). Other fault models have also been used in conjunction with the D-algorithm (Breuer and Friedman, 1976).

The basic idea of path sensitization will first be explained, followed by a description of two path sensitization algorithms and their associated model formulation. The advantages and disadvantages to these two techniques and their models will be pointed out.

We first explain the concepts of line justification and error propagation. A value of $j \varepsilon \{0,1\}$ on the output to gate G is justified by assigning values to the gate inputs which, when applied to G, will result in the value j. For example, the value 1 on the output of an AND gate can be justified by assigning the value 1 to each of the gate inputs. Similarly, the value 0 on the output of an AND gate can be justified by assigning the value 0 to any one or more of the inputs to the gate. Hence, justifying a 0 output to a 2-input AND gate allows three choices. If the inputs are u_1 and u_2, we may choose $u_1 = 1$, $u_2 = 0$; $u_1 = 0$, $u_2 = 1$; or $u_1 = u_2 = 0$.

Adopting the symbolism of (Roth, 1966) the letter D will be used to represent the value of a signal line which has the value 1 in the normal (fault-free) circuit and 0 in the faulty circuit, and D' to represent the value of a signal line which has the value 0 in the fault-free circuit and 1 in the faulty circuit. Hence, if a line has value D, it is stuck-at-0 and D' represents the stuck-at-1 condition.

The error signal D on an input line of gate G is propagated through the gate by assigning values to the other inputs of gate G which will result in a D or D' on the gate output. For example, if an input to an AND gate is assigned the error value D, it will be propagated to the output of the gate only if all other inputs are assigned the value 1. If any other input to the AND gate takes the value 0, the output will be zero and the error value cannot be propagated. To propagate an error signal (either D or D') through an OR gate, all other inputs must be zero; through a NAND gate, all other inputs must be one; through a NOR gate, all other inputs must be zero. Hence, no choice in assignments exists in fault propagation for a given gate type.

The path sensitization procedure for combinational circuits can be described as follows:

Step 1. Select a fault (stuck-at-j, $j \varepsilon \{0,1\}$) and fault site (some signal line, a).

Step 2. Justify the value j' on the signal line a by assigning values to the primary inputs. This justification step may require some choices when assigning values to signal lines and/or primary inputs.

Step 3. Propagate the faulty signal value to a primary output along one path or simultaneously along multiple paths through the circuit. Note that a choice of paths may exist. Propagation of a signal value will require assignments to be made along the chosen path; however, as noted earlier, these assignments are never arbitrary.

Step 4. Justify the assignments made in Step 3 so that they are consistent with the assignments made in Step 2. If a consistent assignment is not possible, then no test exists for the specified fault.

The following example illustrates the path sensitization procedure. Figure 1 is a simple combinational circuit. Assume line x_4 is stuck-at-0. According to Step 2, we wish to justify a 1 on line x_4, and this is done by setting input $u_1 = 1$ and $u_2 = 1$. Next we wish to propagate the fault through the OR gate to line x_6, a primary output. This is done by assigning $x_5 = 0$. Step 4 involves justifying $x_5 = 0$ consistent with the other values already assigned. x_5 will equal zero if $u_2 = 0$, $u_3 = 1$; $u_2 = 1$, $u_3 = 0$; or $u_2 = u_3 = 0$. We must choose $u_2 = 1$, $u_3 = 0$ to be consistent with the assignment made in Step 2. Thus our test consists of the input $u_1 = 1$, $u_2 = 1$, $u_3 = 0$.

It is apparent that there are two items of information about the circuit necessary to perform path sensitization. The topology, i.e., the gate interconnection or network information, is needed to determine the path through the circuit. In addition, the behavior of each gate must be known. For example, the information that an AND gate output is 1 only if both inputs are 1, and 0 only if at least one input is 0, is essential.

Path sensitization procedures differ in the manner in which the topological and behavioral information is stored and manipulated to generate tests. The D-algorithm uses "cubes" to pass behavioral information. An operation called intersection of cubes is performed to establish a chain or path through the circuit. Figure 2 illustrates the cube concept. The inputs to the AND gate are labeled 1 and 2, and the output line is labeled 4. At the right in Figure 2 is a table whose heading has the input/output labels. The first row specifies that the output is 1 when both inputs are 1, and the second and third, that the output is 0

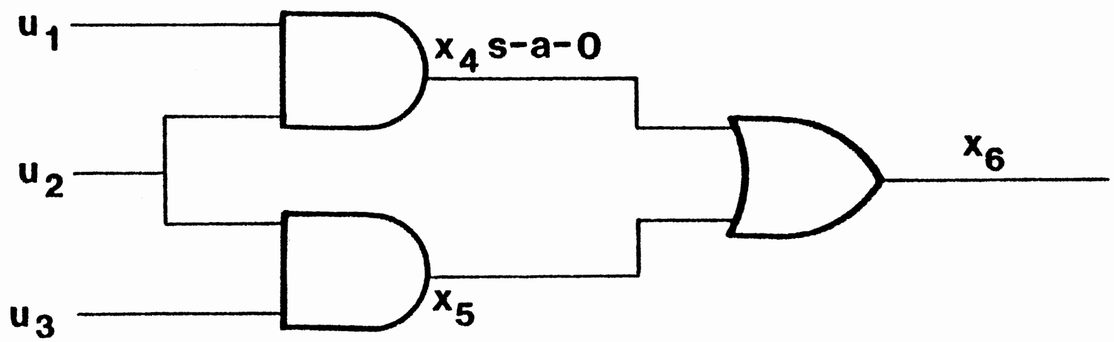


Figure 1. Sample Circuit Illustrating Path Sensitization and Signal Propagation

1	2	4
1	1	1
0		0
	0	0
D	1	D
1	D	D

Figure 2. An AND Gate With Its Singular Cover and D-Cubes

when either input is 0. This is termed the singular cover of the function, and each row in the singular cover is called a cube. Below the singular cover is the error propagation information for the function. These are called D-cubes of a failure. The first cube indicates that a fault on input 1 can be propagated to output 4 if input 2 is 1. These five cubes contain the behavioral information of an AND gate needed to generate a test in a circuit containing an AND gate.

Figure 3 repeats the circuit of Figure 1 showing its singular cover and, within dashed lines, D-cubes. This example is by Roth (1966). The intersection of two cubes determines how circuit conditions specified by the two cubes can be simultaneously satisfied. If the intersection does not exist, the two conditions assign different values to the same line, implying that the conditions are inconsistent. The intersection of cube $A = (a_1, a_2, \ldots, a_n)$ with cube $B = (b_1, b_2, \ldots, b_n)$ designated as $C = (c_1, c_2, \ldots, c_n) = A \div B$ is defined as

$$c_j = a_j$$

if $a_j = b_j$ or if b_j is unspecified. If $a_j \neq b_j$ for some j, then the intersection C does not exist.

The algorithm itself involves intersecting the D-cubes in a trial and error fashion until a chain is formed from the site of the fault to a primary output. Selected node fanin and fanout information is saved as the algorithm progresses so that the topology of the network is progressively learned. Nodes not specified in the D-cube intersecting phase are assigned by intersecting the incomplete test cube with cubes from the singular cover until all lines for which it is possible to assign values have received them. The entire process may require a

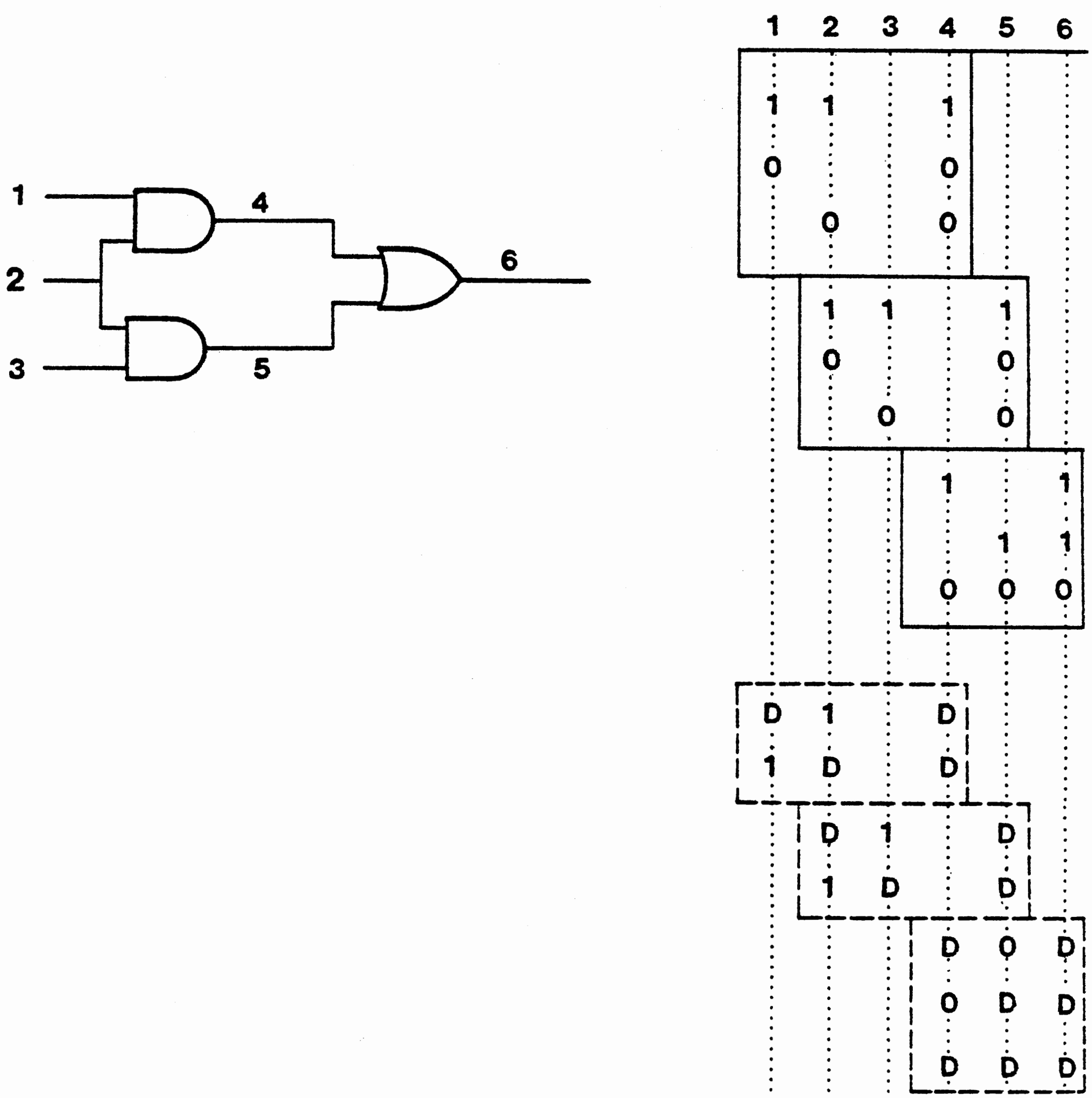


Figure 3. A Simple Combinational Circuit With Its Singular Cover and D-Cubes

great deal of backtracking. A detailed description of the algorithm appears in Breuer and Friedman (1976, pp. 46-49). Detailed examples are also given in the former reference as well as in Roth's original paper (Roth, 1966).

In its most general form, the D-algorithm can always find a test if one exists. If the fault is undetectable, the procedure proves that no test exists.

Schneider (1967) has shown that in some cases it may be necessary to sensitize several paths simultaneously in a reconvergent fanout network to detect a fault. (A network has reconvergent fanout if at least one primary input and/or gate output branches out to two or more gates whose outputs lie on paths through the circuit which later reconverge as inputs to a single gate.) While multiple path sensitization is possible with the D-algorithm, Breuer (1981) asserts that multiple path sensitization cases are rare, and that practical implementations of the D-algorithm are often restricted to single path sensitization in order to reduce computation time.

The cube notation and calculus used by the D-algorithm is adequate with respect to behavioral information and is expressed in a format that is easily stored and manipulated by a computer. The representation of the behavioral information suffers from a lack of compactness, however. The need for so much backtracking in the algorithm execution is an indication that a lack of adequate topological information is available in the circuit model. Paths are discovered and problems such as those encountered in circuits containing reconvergent complemented fanout cannot be avoided due to a lack of topological information. (Reconvergent complemented fanout is defined and discussed in Chapter V.)

A variant of the D-algorithm, critical path sensitization, was introduced by Thomas (1971) and is employed in the stimulus generation section, STIMGN, of the D-LASAR system. The procedure involves first selecting a primary output line and assigning to it a critical 1 (0) value. Using sensitizing cubes, this value is driven back toward the primary inputs as in the justification procedure of the D-algorithm. Whenever a choice exists, one alternative is selected. In order to consider all possibilities, the other choices are reserved and used later by backtracking to the site of the choice and making an alternate assignment. For each choice of assignments, all implications of that choice are made. Each line assignment that is critical to the justification procedure is marked as it is generated, and all critical lines are driven back, all noncritical lines are justified using primitive cubes. When an inconsistency arises, making it impossible to justify a critical value by using a sensitizing cube, one can backtrack and use a primitive cube. This procedure is then repeated for each primary output.

The critical path test generation procedure is outlined in detail with examples in Breuer and Friedman (1976). It may fail to generate a test for a fault which requires multiple path sensitization but detects many faults along a sensitized path in one pass. This later characteristic makes it a fault-independent algorithm.

The D-LASAR system models all circuits in terms of their NAND gate equivalent network. This decreases the amount of behavioral information required, thus compacting that information. The disadvantaqes in modeling everything in terms of NAND gates are twofold. The number of gates in the circuit model can be significantly greater than in the original network. More storage may be required for the network description but less for the cube description. The NAND gate equivalent description has been criticized because it is an abstraction of the original circuit and there may not be a one-to-one correspondence between the stuck-at faults on the signal lines in the model and those of the original circuit.

The significant amount of backtracking and arbitrary assignment of line values when choices arise indicate here, as in the D-algorithm, that not enough topological information is contained in the model. Paths are discovered, as in the D-algorithm, by intersecting cubes to form a long path through the circuit.

Both the D-algorithm and D-LASAR utilize the iterative combinational array to model sequential circuits. In this model, the feedback loops are cut and the circuit reproduced into cells corresponding to an assigned time frame (see Figure 4). The feedback lines form pseudo outputs of time frame j in cell j and pseudo inputs to time frame $j+1$ in cell $j+1$. The test generation procedure results in a sequence of input vectors for each fault. Due to the memory characteristic of sequential circuits, it is desirable to obtain a test sequence that is independent of the initial circuit condition since initialization sequences may be difficult or sometimes impossible to generate. In addition, the presence of a fault in the circuit may invalidate an initialization sequence.

Equation Manipulation

The final test generation procedure to be reviewed is the Boolean difference algorithm. The Boolean difference approach to ATG is a fault-dependent Boolean equation manipulation procedure which utilizes the stuck-at fault model. As a mathematical concept, difference methods can

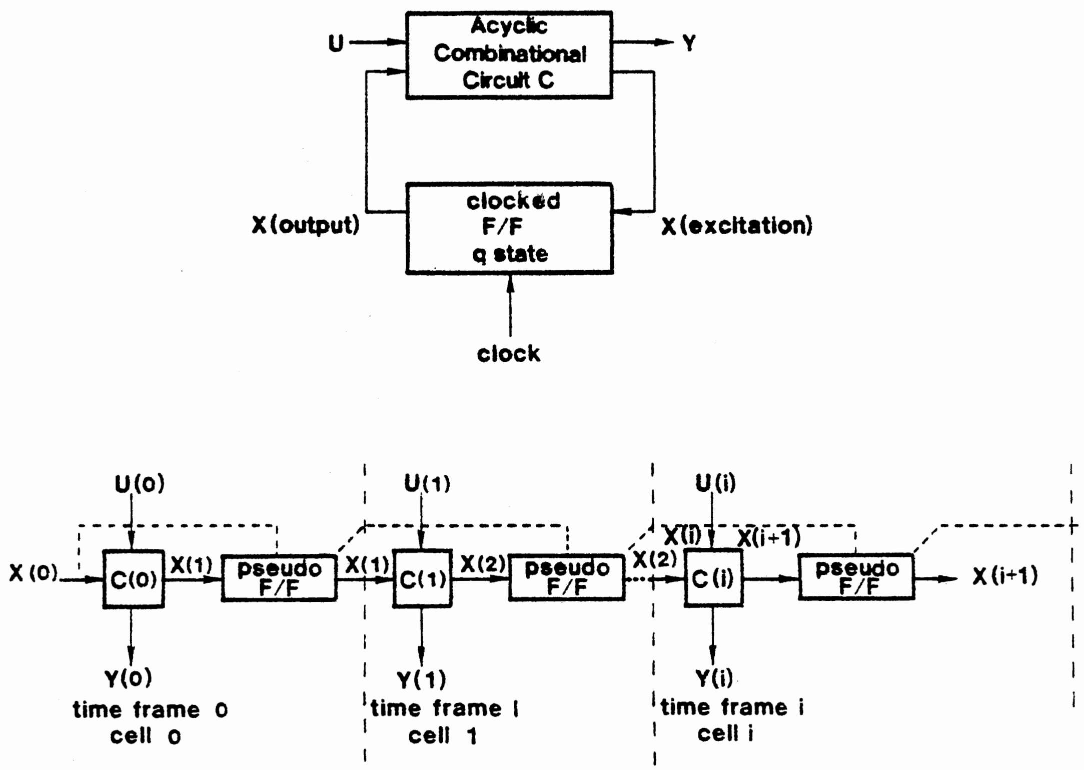


Figure 4. Sequential Circuit Modeled as an Iterative Combinational Array (Breuer and Friedman, 1976)

be traced to G. Boole in 1872. The elegant mathematical formulation of the Boolean calculus was first introduced by Akers (1959). The use of this calculus to derive tests for logic faults in digital circuits was first suggested by Professor E. Stabler to L. W. Bearnson at Syracuse University (Bearnson, 1965). Further development of the idea appeared in a paper by Sellers et al. (1968) and has since undergone extensive development. It is considered to be the classical equation manipulation approach to ATG.

Assume that a combinational circuit with inputs $u_1, u_2, \ldots, u_n$ realizes the function $f(u_1, u_2, \ldots, u_n)$. Let $f_i(0) = f(u_1, \ldots, u_{i-1}, 0, u_{i+1}, \ldots, u_n)$ and $f_i(1) = f(u_1, \ldots, u_{i-1}, 1, u_{i+1}, \ldots, u_n)$. These represent the condition when input u_i is stuck-at-0 and stuck-at-1, respectively. The objective in test generation is to find an input vector $U = (u_1, u_2, \ldots, u_n)$ that causes $f(U)$ and $f_i(0)$ $(f_i(1))$ to take on different values so that the presence of the fault will be detectable at the primary output. Equivalently, we wish to solve the equation $f(U) \oplus f_i(0) = 1$ $(f(U) \oplus f_i(1) = 1)$ for U.

By Shannon's expansion algorithm, we have

$$\begin{aligned} f(U) \oplus f_i(0) &= [u_i f_i(1) + u_i' f_i(0)] \oplus f_i(0) \\ &= [u_i f_i(1) + u_i' f_i(0)]\ f_i'(0) \\ &\quad + f_i(0)[u_i f_i(1) + u_i' f_i(0)]' \\ &= u_i f_i(1)\ f_i'(0) + u_i f_i'(1)\ f_i(0) \\ &= u_i [f_i(1) \oplus f_i(0)] \end{aligned}$$

The factor $f_i(1) \oplus f_i(0)$ is referred to as the Boolean difference of f with respect to u_i and is given the symbol df/du_i. This symbol should not be confused with the derivative notation in the calculus. The test for u_i stuck-at-0 is the input vector (or vectors) U that satisfy the equation

$$u_i \frac{df}{du_i} = 1 .$$

Similarly, it can be shown that a test for u_i stuck-at-1 is the solution of the equation

$$u_i' \frac{df}{du_i} = 1 .$$

The circuit of Figure 1 realizes the Boolean function $f(u_1, u_2, u_3) = u_2(u_1 + u_3)$. The test for u_1 stuck-at-0 is the solution of $u_1[f(0, u_2, u_3) \oplus f(1, u_2, u_3)] = u_1[u_2u_3 \oplus u_2] = 1$ or $u_1u_2u_3' = 1$. The solution is (1,1,0) which tests for u_1 stuck-at-0 (see Table I).

TABLE I

BOOLEAN DIFFERENCE IDENTITIES

$$\frac{df'(U)}{du_i} = \frac{df(U)}{du_i} \tag{1}$$

$$\frac{d[f(U) \cdot g(U)]}{du_i} = f(U) \cdot \frac{dg(U)}{du_i} \oplus g(U) \frac{df(U)}{du_i} \oplus \frac{df(U)}{du_i} \cdot \frac{dg(U)}{du_i} \tag{2}$$

$$\frac{d[f(U) + g(U)]}{du_i} = f'(U) \cdot \frac{dg(U)}{du_i} \oplus g'(U) \frac{df(U)}{du_i} \oplus \frac{df(U)}{du_i} \cdot \frac{dg(U)}{du_i} \tag{3}$$

$$\frac{d[f(U) \oplus g(U)]}{du_i} = \frac{df(U)}{du_i} \oplus \frac{dg(U)}{du_i} \tag{4}$$

The use of the identities listed in Table I reduces the amount of algebraic manipulation required in order to determine df/du_i. If one of the functions, say $g(U)$, is independent of the variable u_i, then $dg/du_i = 0$. When this situation exists, Equations (2) and (3) simplify as follows:

$$\frac{d[f(U) \cdot g(U)]}{du_i} = g(U) \frac{df(U)}{du_i} \tag{2a}$$

$$\frac{d[f(U) + g(U)]}{du_i} = g'(U) \frac{df(U)}{du_i} \tag{3a}$$

In its original formulation, the Boolean difference approach could generate tests for stuck-at faults on primary input lines only. The concept of a chain rule to enable the computation of composite Boolean differences from the partial Boolean differences of the component functions was introduced by Marinos (1971). For example, the circuit of Figure 5 may be thought of as a composite of the functions f and g, where $f = x_1 = u_1u_2$ and $g = x_3 = f + u_3u_4$. To generate a test for x_1 stuck-at-0, we solve

$$x_1 \frac{dx_3}{dx_1} = u_1u_2 \frac{d(x_1 + u_3u_4)}{dx_1} = u_1u_2(u_3' + u_4') = 1 .$$

The tests are (1,1,0,x) and (1,1,x,0), where x represents a "don't care" value.

If each circuit element is considered to be a separate subcircuit, the method resembles the path sensitization procedure. A general form of the chain rule was derived by Chang, Reed, and Banes (1972). Their formula is closely related to the multiple path sensitization concept. Applications of the Boolean difference approach for fault detection in

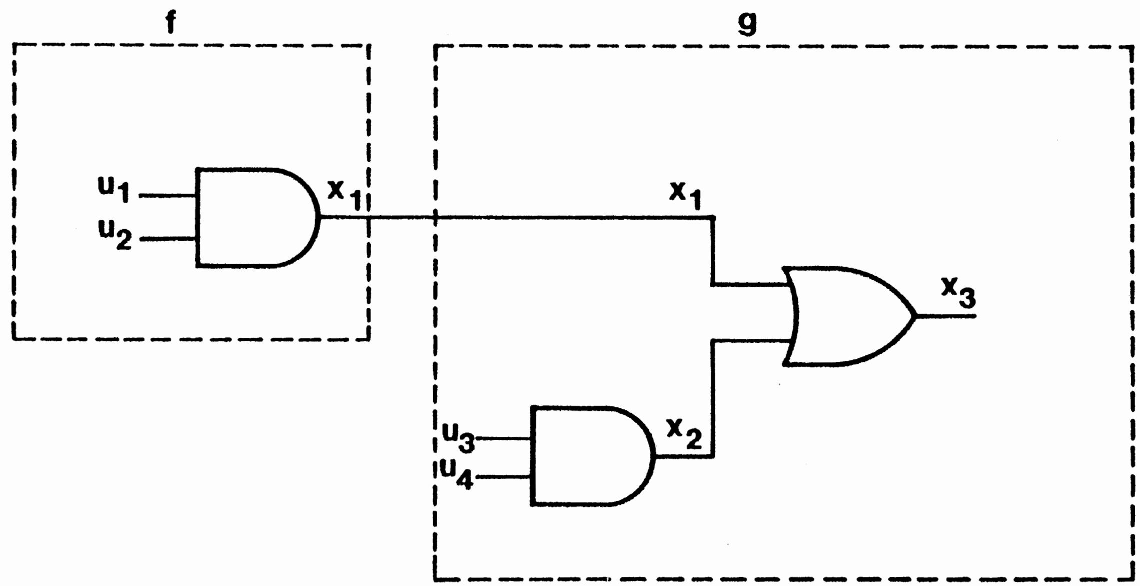


Figure 5. Circuit Viewed as Composite Functions

both asynchronous and synchronous sequential machines exist (Prior and Bennetts, 1974; Hsiao and Chia, 1971).

The Boolean difference approach derives all possible tests which detect a given fault. Since the equation for the circuit is needed, the time and memory requirements for generation of the circuit equations may be excessive for large circuits. Computers are not known to be particularly efficient at manipulation of symbols and simplification of equations.

Summary

The fundamental concepts involved in two basic approaches to automatic test generation have been presented. The path sensitization procedure introduced in Roth's D-algorithm (1966) and later modified for use in D-LASAR (Thomas, 1971) is performed by representing the behavior of a circuit by cubes. A path is traced through the circuit by intersecting the cubes according to the rules of the cube calculus. It was noted that in both instances, the behavioral information was adequately supplied to make the algorithm efficient; however, lack of adequate topological information in the cubical representation led to time consuming backtracking in the algorithms.

The Boolean Difference procedure was shown to be an elegant mathematical concept which generates all possible tests for stuck-type faults. However, since it requires the Boolean equation of the circuit, time and memory requirements for generating the equation for large circuits may be excessive.

CHAPTER III

MODELING DIGITAL SYSTEMS

Introduction

Automatic test generation algorithms based upon a gate level circuit model can benefit from both topological and behavioral information. The efficiency of the algorithm depends, to a large extent, on the mathematical formulation of this information. The mathematical model should be capable of simultaneously describing the gate level interconnection structure and the logical behavior of each individual gate in a concise mathematical representation suitable for easy storage in and manipulation by a computer.

In this chapter, a mathematical model of digital systems is introduced. The model represents a digital system as a set of discrete time Boolean equations expressed in matrix notation. The emphasis of this chapter is to illustrate how the model is formulated from a gate level circuit diagram and to illustrate the representation of behavioral information. The following chapter emphasizes the features of the model pertaining to the network topology and to the machine storage complexity of the model itself.

Notation

A Boolean function is a function f with domain $\{0,1\}^n = \mathcal{B}^n$ and range $\mathcal{B} = \{0,1\}$, where n is an integer and $\mathcal{B}^n$ represents the n-fold

Cartesian product of the set B with itself. A set of m Boolean functions $f_1, f_2, \ldots, f_m$, each of which maps B^n into B, can be represented by a single function f whose domain is B^n and whose range is B^m.

There are exactly four Boolean functions from B into B. These functions are:

Identity	$f_i(x) = x$
Complement (NOT)	$f_c(x) = x'$
Constant Unit	$f_u(x) = 1$
Constant Zero	$f_o(x) = 0$

for $x \varepsilon B$. The notation can be simplified by dropping the f and the parentheses from the functional notation.

Identity	$ix = x$
Complement (NOT)	$cx = x'$
Constant Unit	$ux = 1$
Constant Zero	$ox = 0$.

Two other very important functions are the Boolean AND and Boolean OR functions whose truth tables are exhibited in Table II. These functions are used so frequently that a shorthand notation exists for them: $f_{OR}(x_1, x_2) = x_1 + x_2$, $f_{AND}(x_1, x_2) = x_1 \cdot x_2$. One canonical representation of a Boolean function is the disjunctive normal form (DNF) which is formed by composition of the AND, OR, and NOT functions. Because every Boolean function can be represented through functional composition of the three functions AND, OR, and NOT, these three functions form a basis set for the set of all Boolean functions. Other basis sets exist, e.g., {OR, NOT} and {AND, NOT}. When the two functions AND and NOT are composed, the resulting composition function is called the NAND (NOT

AND) function. Similarly, the NOR (NOT OR) function arises from the composition of the OR and NOT functions. Each of these functions also forms a basis for the set of all Boolean functions.

TABLE II

TRUTH TABLES FOR "AND" AND "OR" FUNCTIONS

x_1	x_2	$f_{AND}(x_1,x_2) = x_1 \cdot x_2$	x_1	x_2	$f_{OR}(x_1,x_2) = x_1 + x_2$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

Let $C = \{i,o,c,u\}$ be the set of functions from B into B and F be the set of all $m \times n$ matrices whose entries are functions from the set C. The elements of F are called transformations from B^n into B^m. In particular for $U = (u_1, u_2, \ldots, u_n)^T \varepsilon B^n$, we define the $m \times n$ transformation matrix A operating on U as

$$AU = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{1m} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} a_{11}\ u_1 + a_{12}\ u_2 + \ldots + a_{1n}\ u_n \\ \cdot \quad \cdot \qquad\qquad\qquad \cdot \\ \cdot \quad \cdot \qquad\qquad\qquad \cdot \\ \cdot \quad \cdot \qquad\qquad\qquad \cdot \\ a_{m1}\ u_1 + a_{m2}\ u_2 + \ldots + a_{mn}\ u_n \end{bmatrix}$$

The set F contains an $(n \times n)$ identity transformation $I_n = (a_{kj})$, where $a_{kj} = i$ when $k = j$ and $a_{kj} = 0$ when $k \neq j$. The identity transformation maps every vector $X\varepsilon B^n$ into itself. The $(m \times n)$ zero transformation, $0_{m,n} = (a_{ij})$, is the matrix, all of whose entries are $o\varepsilon C$. The zero transformation maps every $X\varepsilon B^n$ into an m-vector, all of whose elements are zero.

If A, $B\varepsilon F$ and $X, U\varepsilon B^n$ then $AX + BU\varepsilon B^m$, where addition is understood to be the Boolean OR operation on vectors in B^m.

A transformation $T\varepsilon F$ is, in general, nonlinear. This can be easily proven by exhibiting a counter example. Let $T:B^3 \rightarrow B^2$ be the transformation $T(u_1, u_2, u_3)^T = (cu_1, o)^T$.

Then $T = \begin{bmatrix} c & o & o \\ o & o & o \end{bmatrix}$ in matrix form. For α, $\beta\varepsilon B$,

$$T(\alpha U + \beta V) = \begin{bmatrix} c & o & o \\ o & o & o \end{bmatrix} \begin{bmatrix} \alpha u_1 + \beta v_1 \\ \alpha u_2 + \beta v_2 \\ \alpha u_3 + \beta v_3 \end{bmatrix} = \begin{bmatrix} c(\alpha u_1 + \beta v_1) \\ o \end{bmatrix} = \begin{bmatrix} (c\alpha + cu_1)(c\beta + cv_1) \\ o \end{bmatrix}$$

$$\neq \alpha TU + \beta TV .$$

Combinational Circuit Model

Logic circuits physically realize Boolean functions through composition of functions from the basis set {AND, OR, NOT}. The gate level logic diagram of a combinational circuit describes the interconnection of basic logic gates which represent the Boolean functions AND, OR, NOT, NAND, and NOR. Given a gate level logic diagram, we adopt the following line labeling convention: Primary inputs are labeled $u_1, u_2, \ldots, u_n$, and the internal lines are labeled x_j, where the subscript j must be greater than n and greater than the subscript of any line label on any path between the primary inputs and the line x_j. Figure 6 is a labeled

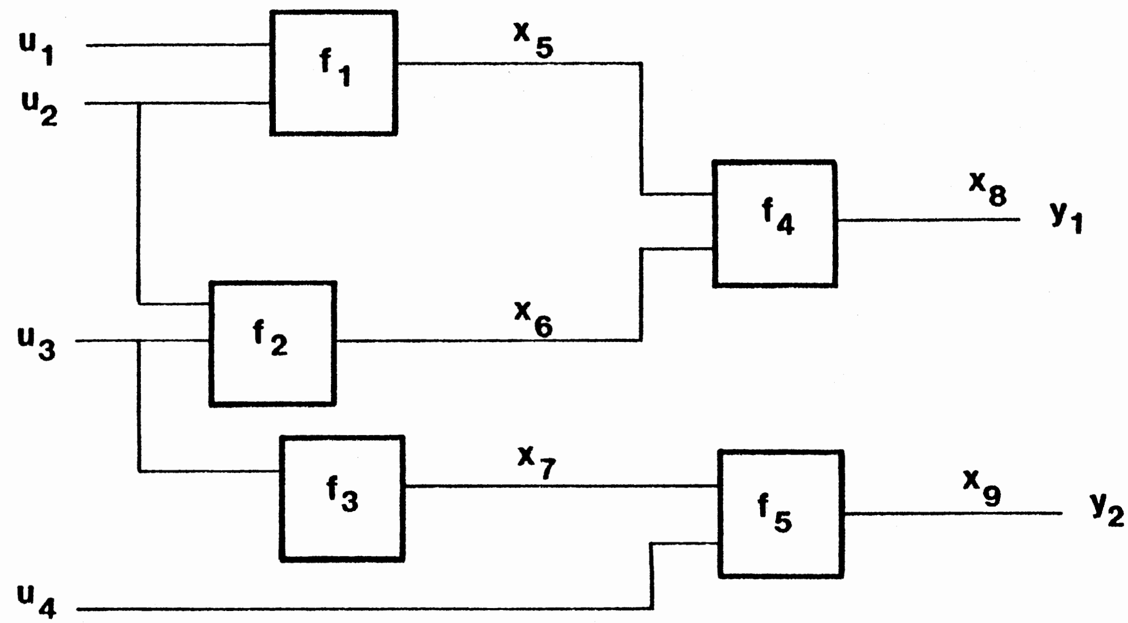


Figure 6. Labeled Generic Logic Diagram

generic logic diagram where f_i represents one of the Boolean functions AND, OR, NOT, NAND, or NOR.

The logic diagram can be thought of as a graph of the chain of Boolean function compositions. For example, in Figure 6

$$x_1 = iu_1$$
$$x_2 = iu_2$$
$$x_3 = iu_3$$
$$x_4 = iu_4$$
$$x_5 = f_1(x_1, x_2)$$
$$x_6 = f_2(x_2, x_3)$$
$$x_7 = f_3(x_3)$$
$$x_8 = f_4(x_5, x_6)$$
$$x_9 = f_5(x_4, x_7)$$

where x_1 through x_4 represent data steps, and x_5 through x_9 represent computation steps. This is called a 9-step chain. (See Savage [1976] for a definition of k-step computation chains.)

If we restrict the logic diagram to contain only gates which represent the Boolean functions NAND, OR, and NOT, the computation chain takes on a linear-looking form. The circuit in Figure 7 contains only NAND, NOT, and OR gates. The original circuit was an AND,OR circuit. The output lines of NOT gates inserted during modeling are not labeled. The computation chain of the modeled circuit is as follows:

$$\left.\begin{array}{l} x_1 = iu_1 \\ x_2 = iu_2 \\ x_3 = iu_3 \end{array}\right\} \quad \text{DATA STEPS}$$

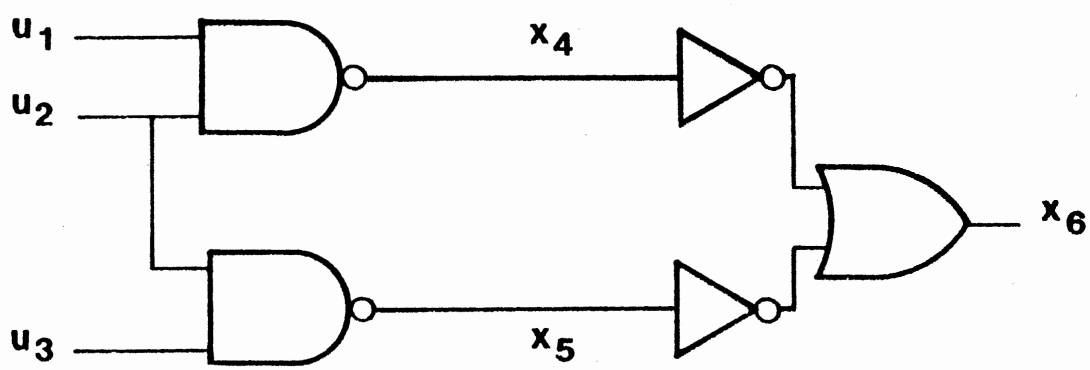


Figure 7. A NAND, OR, NOT Circuit

$$\left.\begin{aligned} x_4 &= cx_1 + cx_2 \\ x_5 &= cx_2 + cx_3 \\ x_6 &= cx_4 + cx_5 \end{aligned}\right\} \quad \text{COMPUTATION STEPS}$$

In matrix notation the set of equations is written:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} o & o & o & o & o & o \\ o & o & o & o & o & o \\ o & o & o & o & o & o \\ c & c & o & o & o & o \\ o & c & c & o & o & o \\ o & o & o & c & c & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} i & o & o \\ o & i & o \\ o & o & i \\ o & o & o \\ o & o & o \\ o & o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$[y] = [o\ o\ o\ o\ o\ i] \quad X$$

where the entries in the matrices are functions from the set $\mathcal{C}$ = $\{i, o, c, u\}$ of Boolean functions of one variable.

In general, the equations describing the behavior of a digital combinational logic circuit with n primary inputs $u_1, u_2, \ldots, u_n$; p logic gates of the type AND, OR, NOR, NAND, NOT, whose outputs are labeled $x_{n+1}, x_{n+2}, \ldots, x_{n+p}$; and m primary outputs $y_1, y_2, \ldots, y_m$ can be expressed in the form

$$X = AX + BU$$

$$Y = DX$$

where $X = (x_1, x_2, \ldots, x_{n+p})^T \,\varepsilon \mathcal{B}^{n+p}$, $U = (u_1, u_2, \ldots, u_n)^T \,\varepsilon \mathcal{B}^n$, and $Y = (y_1, y_2, \ldots, y_m)^T \,\varepsilon \mathcal{B}^m$. The matrix $A \varepsilon \mathcal{F}$ is an (n+p, n+p) matrix which can be partitioned into submatrices.

$$A = \left[\begin{array}{c|c} 0_{n,n} & 0_{n,p} \\ \hline A_{1\,p,n} & A_{2\,p,p} \end{array}\right] .$$

The matrix $B \varepsilon \mathcal{F}$, a (n+p,n) matrix, can also be partitioned into sub-matrices.

$$B = \begin{bmatrix} I_{n,n} \\ \\ 0_{p,n} \end{bmatrix} .$$

$D \varepsilon \mathcal{F}$ is an (m,n+p) matrix. Because of the labeling convention, the A matrix is lower triangular, as is the matrix A_2.

During modeling, all circuit gates are represented in terms of NAND, OR, and NOT gates to achieve the linear-looking equations. Three questions arise:

1. How difficult is it to model a circuit in terms of these gates only?
2. What relationship does the signal line in the model have to the actual gate level circuit description?
3. What effect does modeling with NAND, OR, and NOT gates have on test generation for stuck-at faults?

The first two questions will be addressed simultaneously. Figure 8 shows how easily each circuit element can be represented in terms of the NAND, OR, NOT gates. There is a one-to-one correspondence between the signal lines in the standard representation and the model representation since the output to inserted inverters are not labeled.

A test for the output of an AND (OR) gate stuck-at-1 is also a test for the output of a NAND (NOR) gate stuck-at-0. When an ATG algorithm tries to generate tests for all stuck-at-1 and stuck-at-0 faults in the

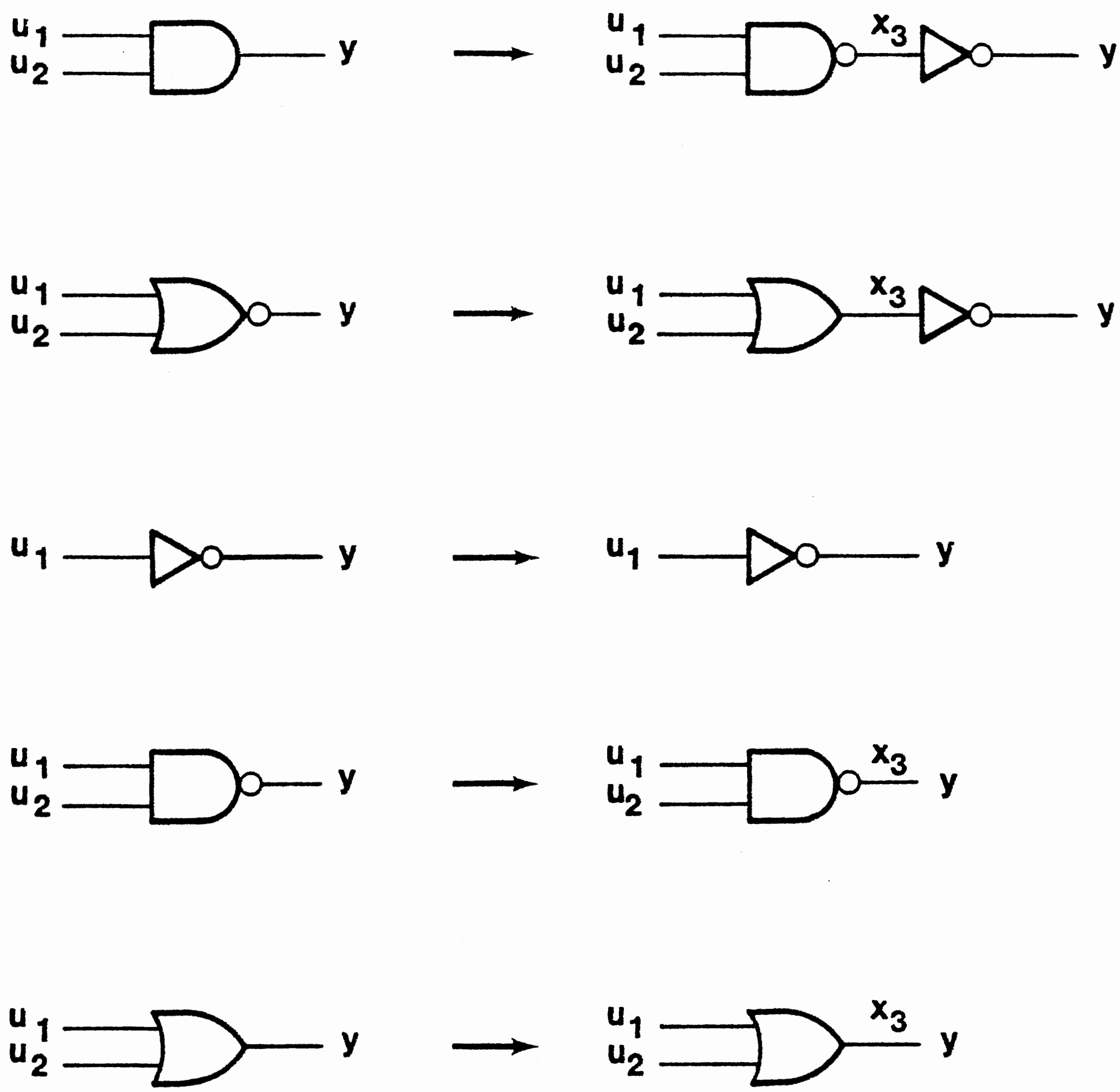


Figure 8. Model Representation of Logic Gates

circuit model, the test derived will also test for the stuck-at-faults in the original circuit. Knowledge of the fact that ANDs are modeled with NANDs followed by inverters and NORs as ORs followed by inverters, and knowledge of the above facts concerning the relationship between stuck-at faults in the modeled circuit and the actual circuit enables a straightforward fault location and diagnosis procedure.

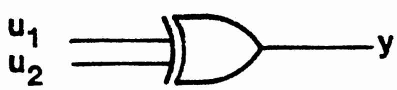
Figure 9 illustrates the modeling process using an exclusive-OR (XOR) gate. Two ways to model it are shown. The matrix equation associated with each model appears on the right.

To further illustrate the modeling procedure for combinational circuits, Figure 10 shows a gate level diagram of a circuit and the representation from which a transformation matrix can be written. The matrix equation is

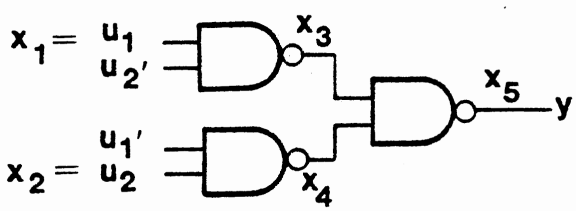
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} o & o & o & o & o & o \\ o & o & o & o & o & o \\ o & o & o & o & o & o \\ c & c & o & o & o & o \\ o & i & i & o & o & o \\ o & o & o & c & c & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} i & o & o \\ o & i & o \\ o & o & i \\ o & o & o \\ o & o & o \\ o & o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$[y] = [o \ o \ o \ o \ o \ i] \quad X$$

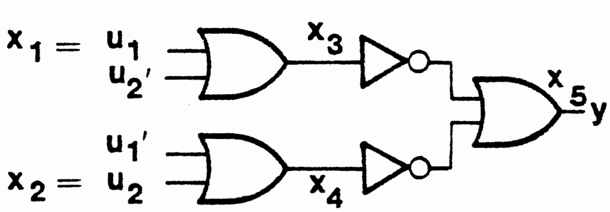
The modeling of all logic gates in terms of NAND, OR, and NOT gates is an intermediate step between the logic circuit described with AND, OR, NOT, NAND, NOR, and XOR gates and that proposed by Thomas (1971) of using only NAND gates. While the NAND gate model can also be represented in the matrix formulation, the complexity of modeling is slightly



$$y = u_1 \oplus u_2$$

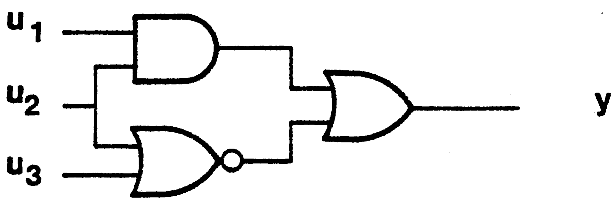


$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} i & o & o & o & o \\ o & i & o & o & o \\ c & i & o & o & o \\ i & c & o & o & o \\ o & o & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

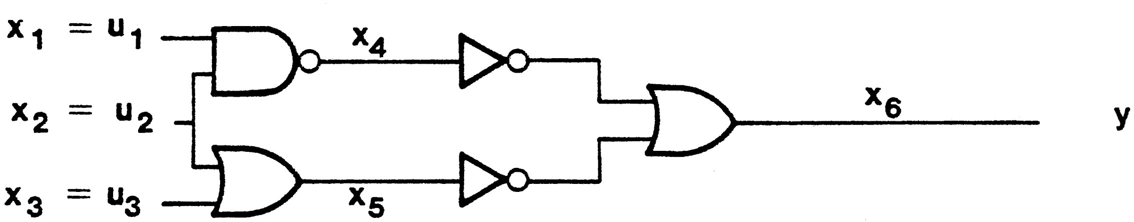


$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} i & o & o & o & o \\ o & i & o & o & o \\ i & c & o & o & o \\ c & i & o & o & o \\ o & o & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

Figure 9. Exclusive-OR (XOR) Representations



(a) Gate Level Combinational Circuit Diagram



(b) Gate Level Transformation

Figure 10. Combinational Circuit Modeling

larger. Figure 11 shows the OR gate and NOT (inverter) modeled as NAND gates. The matrix representations, however, have the same dimensions.

It is possible to use the idea of computation chains to model the combinational circuit and omit the data steps. When this is done, a smaller-dimensioned representation is obtained; however, it is not as useful in ATG as the one presented. For completeness, however, Figure 12 illustrates the labeling procedure and matrix representation for the same circuit as Figure 10. Note that the matrix operating on the U vector in Figure 12 is the same as the submatrix A_1 in the representation for Figure 10.

Before introducing the sequential circuit model, one other interesting observation is made. It is possible to represent a Boolean function in the matrix representation, independent of a gate level diagram. For example, if

$$f(u_1, u_2, u_3) = u_1 u_2 u_3' + u_2' u_3 ,$$

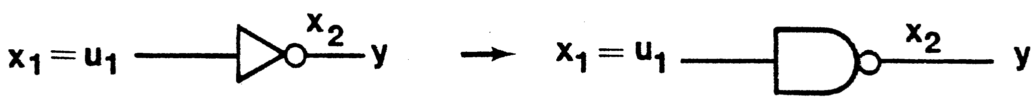
let

$$\left.\begin{aligned} x_1 &= u_1 \\ x_2 &= u_2 \\ x_3 &= u_3 \end{aligned}\right\} \quad \text{DATA STEPS}$$

$$\left.\begin{aligned} x_4 &= cx_1 + cx_2 + ix_3 \\ x_5 &= ix_2 + cx_3 \\ x_6 &= cx_4 + cx_5 \end{aligned}\right\} \quad \text{COMPUTATION STEPS}$$

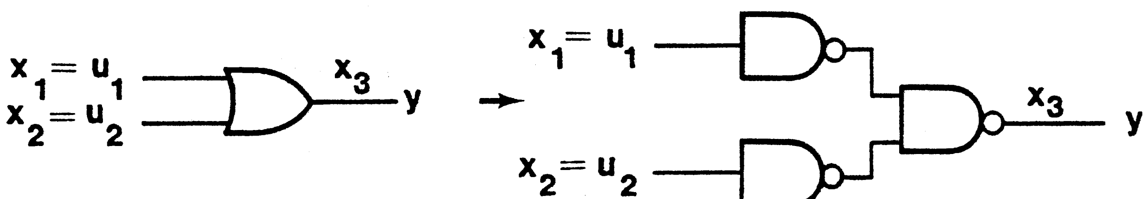
and

$$y = ix_6 .$$



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} o & o \\ c & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} i \\ o \end{bmatrix} \begin{bmatrix} u_1 \end{bmatrix}$$

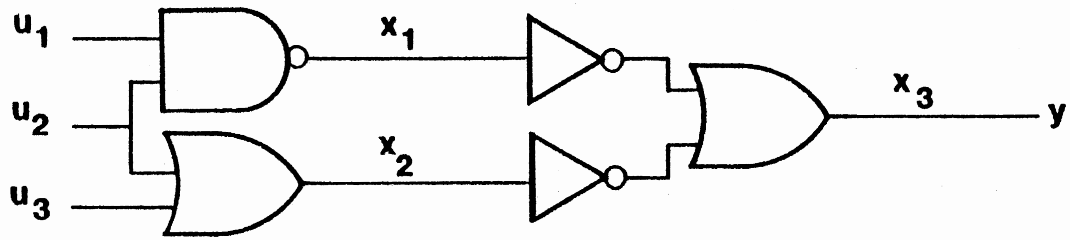
$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} o & i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} o & o & o \\ o & o & o \\ i & i & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} i & o \\ o & i \\ o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} o & o & i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Figure 11. NOT and OR Gates Represented in Terms of NAND Gates



$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} o & o & o \\ o & o & o \\ c & c & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} c & c & o \\ o & i & i \\ o & o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} o & o & i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Figure 12. Compact Matrix Formulation

In matrix form

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} o & o & o & o & o & o \\ o & o & o & o & o & o \\ o & o & o & o & o & o \\ c & c & i & o & o & o \\ o & i & c & o & o & o \\ o & o & o & c & c & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} i & o & o \\ o & i & o \\ o & o & i \\ o & o & o \\ o & o & o \\ o & o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$[y] = [o\ o\ o\ o\ o\ i] \quad X$$

Sequential Circuit Model

The sequential circuit will be modeled as an iterative combinational array (Breuer and Friedman, 1976). The matrix formulation achieves a concise representation for this model. The delay symbol pictured in Figure 13 will be incorporated into the gate level diagram in feedback loops to denote the time frames (not to be confused with clock pulses in synchronous sequential circuits or with actual circuit delay).

The computation chain for the circuit of Figure 13b is

$$x_1(t) = u_1(t)$$

$$x_2(t) = u_2(t)$$

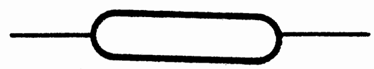
$$x_3(t) = cx_1(t) + cx_4(t-1)$$

$$x_4(t) = cx_2(t) + cx_3(t-1)$$

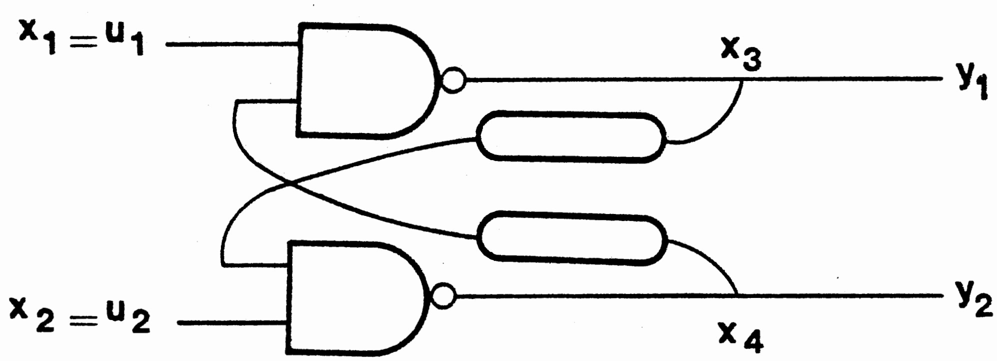
The computation chain is shown in matrix form on page 41. In general, a sequential circuit can be represented in the form

$$X(t) = AX(t) + BU(t) + CX(t-1)$$

$$Y(t) = DX(t)$$



(a) Delay Symbol



(b) Delay Symbol in Feedback Loop

Figure 13. Sequential Circuit Modeling

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} o & o & o & o \\ o & o & o & o \\ c & o & o & o \\ o & c & o & o \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} i & o \\ o & i \\ o & o \\ o & o \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} + \begin{bmatrix} o & o & o & o \\ o & o & o & o \\ o & o & o & c \\ o & o & c & o \end{bmatrix} \begin{bmatrix} x_1(t-1) \\ x_2(t-1) \\ x_3(t-1) \\ x_4(t-1) \end{bmatrix}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} o & o & i & o \\ o & o & o & i \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

where A, B, C, and DεF, t is an integer time step, U(t) εB^n, Y(t) εB^m. A and B can be partitioned as in the combinational case and the matrix C can be partitioned

$$C = \left[\begin{array}{c|c} 0_{n,n} & 0_{n,p} \\ \hline C_{1p,n} & C_{2p,p} \end{array}\right]$$

Figure 14 is a labeled gate level diagram of a clocked D-flip flop. The matrix representation for this synchronous sequential flip flop is shown on page 44.

Summary

Mathematical matrix representations which describe the behavior of digital systems have been derived. The gate level logic diagram is represented in terms of NAND, OR, and NOT gates and a k-step computation chain is formed. The coefficients of each variable in the computation chain is a Boolean function from the set C = {o,i,c,u}. Sequential circuits are represented as iterative combinational arrays and represented in the matrix format through the introductions of a delay element in each feedback loop. Mathematically, the delay is represented by an integer, t, which corresponds to the iteration number.

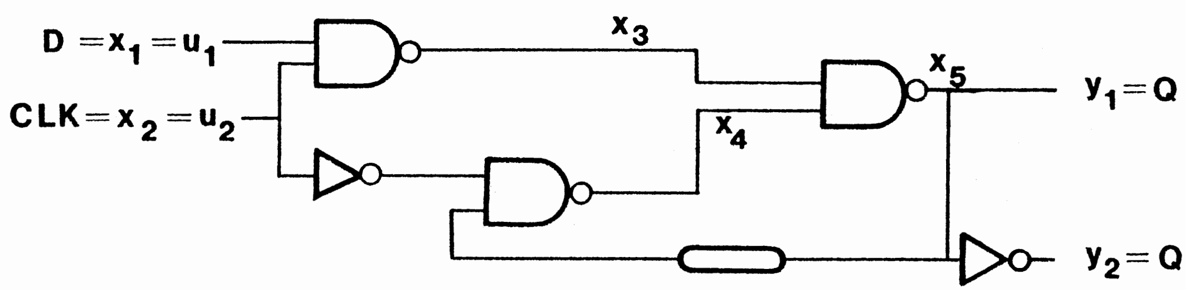


Figure 14. Clocked D-Flip Flop

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \end{bmatrix} = \begin{bmatrix} o & o & o & o & o \\ o & o & o & o & o \\ c & c & o & o & o \\ o & i & o & o & o \\ o & o & c & c & o \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \end{bmatrix} + \begin{bmatrix} i & o \\ o & i \\ o & o \\ o & o \\ o & o \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} + \begin{bmatrix} o & o & o & o & o \\ o & o & o & o & o \\ o & o & o & o & o \\ o & o & o & o & c \\ o & o & o & o & o \end{bmatrix} \begin{bmatrix} x_1(t-1) \\ x_2(t-1) \\ x_3(t-1) \\ x_4(t-1) \\ x_5(t-1) \end{bmatrix}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} o & o & o & o & i \\ o & o & o & o & c \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \end{bmatrix}$$

CHAPTER IV

MODEL CHARACTERISTICS AND EXTENSIONS

Introduction

The logical behavior of digital systems is a dominant feature of digital circuits when generating tests for logical faults. Because it is necessary in automatic test generation to derive an input to the circuit which allows one to observe a fault on an internal signal line at a primary output, most fault-dependent algorithms have taken the path sensitization approach. Locating and tracing specific paths through the circuit requires some topological information. Both the logic behavior and topological information must be presented in a concise, efficient manner that is amenable to computer storage and manipulation.

Characteristics of the matrix model of digital circuits as they relate to the topology of the system will be given in this chapter. Other features of the circuit contained in the matrix description will be pointed out, and the relationship between the size of the describing matrix and circuit size will be presented. The original model will be extended to include variables associated with each fork in a fanout path in the circuit so that complete coverage of single permanent stuck-at faults can be achieved.

Combinational Circuit Topology

A combinational circuit can be thought of as a directed graph whose vertices consist of the primary inputs and logic gates and whose edges are represented by the signal lines. A directed graph corresponding to Figure 10(a) is given in Figure 15. The adjacency matrix associated with a directed graph G having n vertices and no parallel edges is an $n \times n$ binary matrix $A_G = (a_{ij})$ such that

a_{ij} = 1 if there is an edge directed from the ith vertex to the jth vertex, and

= 0 otherwise

(Deo, 1974). The adjacency graph for Figure 15 is

$$A_G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ G_1 \\ G_2 \\ G_3 \end{array} \begin{array}{c} \begin{array}{cccccc} u_1 & u_2 & u_3 & G_1 & G_2 & G_3 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

It can be seen by inspection that the transpose of A_G and the A matrix for Figure 10 have corresponding entries. In particular, every nonzero entry in A corresponds to a nonzero entry in the transpose of A_G. This direct correspondence indicates that the network connection (topological) information is contained in the A matrix of the model. Since the network interconnection information is present in A, the fan-in (fanout) of the individual vertices (primary inputs and logic gates) can be computed by counting the number of nonzero entries in the row (column) associated with that vertex.

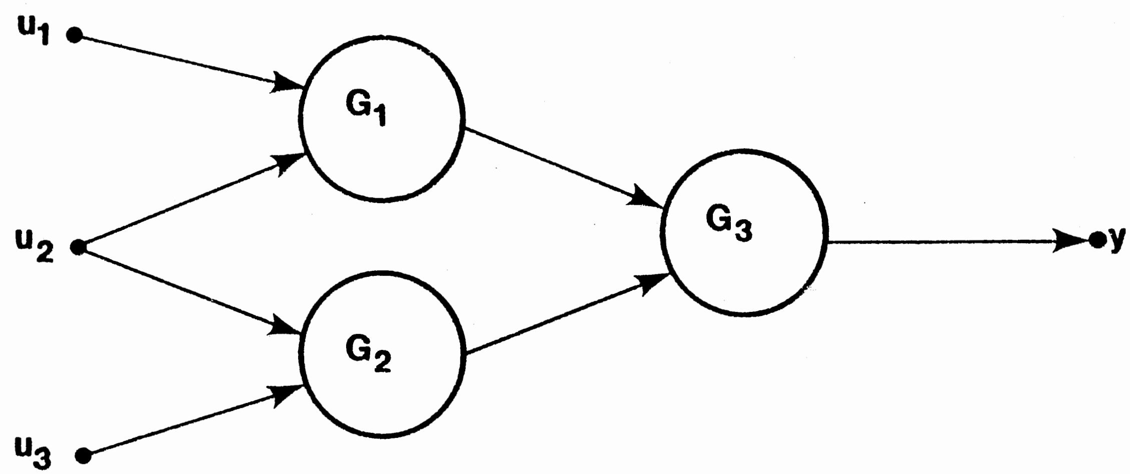


Figure 15. Directed Graph of Figure 10(a)

In particular, define r_i equal to the number of nonzero entries in row i of the A matrix and k_j equal to the number of nonzero entries in column j of the matrix A. Then r_i is the fanin of the gate with output labeled x_i, and k_j is the fanout of the gate with output labeled x_j or of a primary input u_j. The vectors $R = (r_i)$ and $K = (k_j)^T$ will be referred to as the fanin and fanout vectors, respectively.

The fanin vector for the network of Figure 15 is $R = (0,0,0,2,2,2,2)^T$. The fanout vector is $K = (1,2,1,1,1,0)^T$.

Combinational Circuit Matrix Complexity

Define the complexity, γ, of a matrix to be the number of nonzero elements in the matrix. In the adjacency matrix of a graph, G, the integer γ equals the number of edges in G. The matrix parameter γ is related to the amount of computer storage necessary to represent the matrix.

For conciseness, the system of equations

$$X = AX + BU$$

will be rewritten as

$$X = M \begin{bmatrix} U \\ X_p \end{bmatrix}$$

where

$$M = \left[\begin{array}{c|c} I_n & 0 \\ \hline A_1 & A_2 \end{array}\right]$$

and

$$X_p = (x_{n+1}, x_{n+2}, \ldots, x_p) .$$

This formulation takes advantage of the fact that $x_i = u_i$ for $i = 1, 2, \ldots, n$. For example, for the circuit of Figure 10,

$$M = \begin{bmatrix} i & o & o & o & o & o \\ o & i & o & o & o & o \\ o & o & i & o & o & o \\ c & c & o & o & o & o \\ o & i & i & o & o & o \\ o & o & o & c & c & o \end{bmatrix} .$$

The matrix M will be referred to as the defining matrix for the system since it contains both the behavioral and topological information. It can be easily seen that the number of nonzero elements in M

$$\gamma = n + \sum_{n+1}^{n+p} r_i = n + \sum_{j=1}^{n+p} k_j .$$

The lower limit of the first summation is n+1 because the first n rows of A contain only the null element o. The complexity measure, γ, corresponds to the number of primary inputs plus the sum of all fanin (fanout) connections in the circuit including primary outputs. For the circuit of Figure 10, $\gamma = 3 + 6 = 9$. The upper limit on γ is $\gamma_{max} = n + \min \{p \cdot (r_i)_{max}, (n+p)(k_j)_{max}\}$.

A measure of the sparsity of a matrix is given by its density. The density is defined to be the number of nonnull elements divided by the total number of matrix elements. The density of M is

$$d = \frac{\gamma}{(n+p)^2} .$$

The density of the defining matrix M for Figure 10 is $d = 9/36 = 0.25$.

Model Extensions

To completely test a combinational circuit for single permanent stuck-type faults, fanout on primary input signals and at the output of internal gates must be considered. Figure 16, an example from Breuer and Friedman (1976), illustrates the necessity to differentiate between the different forks on a signal path. The input signal u_2 fans out to gates G_1 and G_2. This signal may appear normal at one gate and faulty on the other. If u_2 is stuck-at-1, then both a and b are also 1 and the output y becomes $u_2 + u_3$. However, the fault in which the input lead a to G_1 is stuck-at-1 but the signal b still corresponds to u_2 results in the output $y = u_1 + u_2u_3$. Similarly, if the input lead b to G_2 is stuck-at-1 while signal a still corresponds to u_2, $y = u_1u_2 + u_3$.

This situation can be included in the model by labeling each fork in a fanout path. With this expansion, note that p is no longer equal to the number of gates in the circuit, but is equal to the number of gates plus the number of newly labeled forks. Figure 17 illustrates the labeling procedure for the circuit in Figure 16. The computation chain becomes

$$
\begin{aligned}
x_1 &= u_1 \\
x_2 &= u_2 \\
x_3 &= u_3 \\
x_4 &= x_2 \\
x_5 &= x_2 \\
x_6 &= cx_1 + cx_4 \\
x_7 &= cx_3 + cx_5 \\
x_8 &= cx_6 + cx_7 \quad .
\end{aligned}
$$

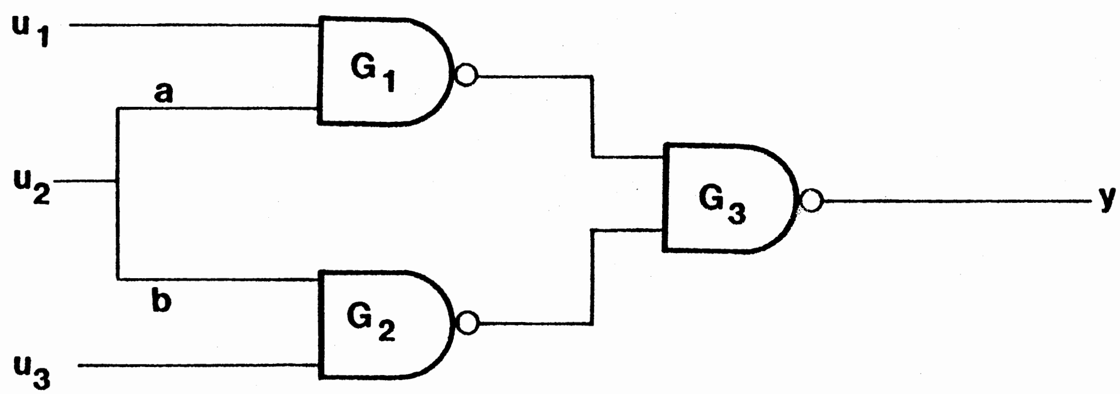


Figure 16. Fanout of a Circuit Input (Breuer and Friedman, 1976)

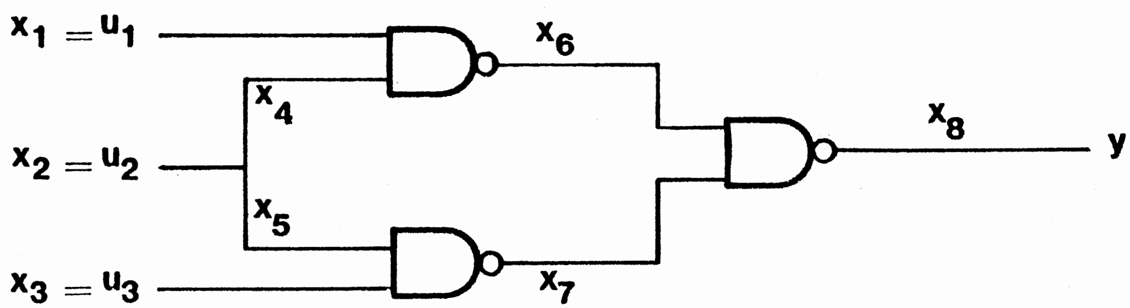


Figure 17. Labeling Extended to Forks in a Fanout Path

In matrix form

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} i & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ o & o & i & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ c & o & o & c & o & o & o & o \\ o & o & c & o & c & o & o & o \\ o & o & o & o & o & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

$$[y] = [o\ o\ o\ o\ o\ o\ o\ i] \quad X$$

The dimension of M is increased by 2 rows and 2 columns, the density of M is 0.17 compared to $d \doteq 0.25$ before the extension, and $\gamma = 11$ compared to $\gamma = 9$ prior to the extension. It is expected that when more details of the circuit are modeled, the complexity of the model will increase.

The fanout vector $K = (1,2,1,1,1,1,1,1,0)^T$ indicates explicitly that $x_2 = u_2$ is the line which forks. In general, when $k_j > 1$, the defining matrix will contain $\sum_{j=1}^{n+p} k_j$, more elements than the nonextended matrix. The maximum increase in complexity, Δ_{max}, due to extending the model is equal to $(k_j)_{max}$ times the number of gates plus primary inputs. The maximum complexity of the extended model is equal to the maximum complexity of the unextended model plus Δ_{max}. This quantity is linear in the number of gates plus primary inputs.

Sequential Circuit Topology

The adjacency matrix for the circuit in Figure 13 (not including the delay symbols as vertices) is

$$A_G = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In the model, topological (directed interconnections) information is contained in both A and C. The entries in the transpose of A_G correspond to entries in A + C, where addition in C is defined by the table:

+	o	i	c	u
o	o	i	c	u
i	i	i	u	u
c	c	u	c	u
u	u	u	u	u

and matrix addition in F is defined as in ordinary matrix algebra, i.e., $A + B = (a_{ij}) + (b_{ij}) = (a_{ij} + b_{ij})$.

In particular, A + C for the model of Figure 13 is given by

$$A + C = \begin{bmatrix} o & o & o & o \\ o & o & o & o \\ c & o & o & c \\ o & c & c & o \end{bmatrix}$$

For sequential circuits, the fanin (fanout) vectors are found by counting the number of nonzero elements in the rows (columns) of A + C. The fanin vector for the above example is $R = (0,0,2,2)^T$ and the fanout vector is $K = (1,1.1,1)^T$.

Sequential Circuit Matrix Complexity

The defining matrix for a sequential circuit is defined to be

$$M = \left[\begin{array}{c:c:c} I_n & 0 & 0 \\ \hdashline A_1 & A_2 & C_2 \end{array}\right]$$

where the submatrices A_1, A_2, and C_2 are submatrices of A and C. The general equation can now be expressed as

$$X(t) = M \begin{bmatrix} U \\ X_p(t) \\ X_p(t-1) \end{bmatrix}$$

where $X_p = (x_{n+1}, x_{n+2}, \ldots, x_{n+p})$.

For the circuit of Figure 14,

$$M = \left[\begin{array}{cc:ccc:ccc} i & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ \hdashline c & c & o & o & o & o & o & o \\ o & i & o & o & o & o & o & c \\ o & o & c & c & o & o & o & o \end{array}\right]$$

and the equation is written as

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \end{bmatrix} = \begin{bmatrix} i & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ c & c & o & o & o & o & o & o \\ o & i & o & o & o & o & o & c \\ o & o & c & c & o & o & o & o \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_3(t-1) \\ x_4(t-1) \\ x_5(t-1) \end{bmatrix}$$

For the defining matrix, M, the complexity of the matrix is defined identically as for the combinational defining matrix. The complexity of M above is $\gamma = 2 + 6 = 8$. The density is again defined as the number of nonzero elements divided by the total number of elements in the matrix. In general, for a sequential circuit,

$$d = \frac{\gamma}{(n+p)(n+2p)} .$$

In situations where lines fan out in a sequential circuit, the model can be extended as in the combinational case.

Summary

The matrix representation for both combinational and sequential circuits contains the network interconnection information such as node adjacency, fanin, and fanout. The number of elements in the defining matrix of a system corresponds to the number of primary inputs plus the sum of all fanin (fanout) connections in the circuit, including primary outputs and feedback lines in sequential circuits.

CHAPTER V

A MATRIX APPROACH TO AUTOMATIC TEST GENERATION

Introduction

In this chapter, the matrix model of digital systems forms the framework in which the path sensitization approach of Roth is carried out. Behavioral and topological features of the model enable tests to be generated for several faults at once. When choices arise as to line assignments during path sensitization, the model contains enough information to make the assignments intelligently, thus reducing the requirement for backtracking. Self-initializing test sequences for sequential circuits can be generated and untestable faults flagged.

The application of the model in the context of automatic test generation using the Boolean difference method is also exhibited. An algorithm for deriving Boolean equations for each internal line in terms of primary inputs is presented. The Boolean difference of sums of complemented and uncomplemented variables is very straightforward. Because the matrix model represents lines in this manner, differences are quickly and easily calculated when needed, thus saving the computer storage space once necessary for storing the differences.

Path Sensitization Using the Matrix Model

The matrix model introduced in Chapters III and IV represents each internal line in terms of the sum (OR) of complemented and/or uncomplemented variables. The variables may be either primary inputs or internal lines. The flow of behavioral information proceeds through the matrix model from inputs to outputs by data steps followed by computation steps.

Line justification and error propagation for the Boolean OR operation is reviewed again at this point for reference. If $x_4 = x_2 + x_3'$, then $x_4 = 1$ can be justified by the assignment $x_2 = 1$ or $x_3 = 0$. To justify $x_4 = 0$, the assignment $x_2 = 0$ and $x_3 = 1$ is required. To propagate the fault $x_2 = D$ to line x_4 requires that $x_3' = 0$ $(x_3 = 1)$. To propagate the fault $x_3 = D$ to line x_4 $(x_4 = D')$, it is necessary that $x_2 = 0$.

In general, to justify a 1, at least one uncomplemented summand must be 1, or at least one complemented variable in the sum must be 0. To justify a 0, all uncomplemented summands must be 0 and all complemented summands, 1. To propagate an error signal (D or D'), each uncomplemented variable in the sum must be 0 and each complemented variable, 1.

The path sensitization approach outlined in Chapter II involved first deciding on a fault and fault site. The fault-free value of the faulty variable is immediately justified at a primary input and the error signal is then propagated to a primary output. The last step involves the consistent justification of line values assigned during the justification and propagation steps.

The manner in which this procedure is carried out within the matrix model can best be understood by illustration. Examples showing the basic path sensitization method for combinational circuits, multiple path sensitization, the undetectable fault case, and self-initialization procedures for sequential circuits are presented. A flow diagram generalizing the procedure follows the illustrations.

The combinational circuit of Figure 18 has matrix representation:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \end{bmatrix} = \begin{bmatrix} i & o & o & o & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o & o & o & o \\ o & o & i & o & o & o & o & o & o & o & o \\ c & c & o & o & o & o & o & o & o & o & o \\ o & c & c & o & o & o & o & o & o & o & o \\ o & o & o & o & i & o & o & o & o & o & o \\ o & o & o & o & i & o & o & o & o & o & o \\ c & o & c & o & o & o & o & o & o & o & o \\ o & c & o & c & o & c & o & o & o & o & o \\ o & o & c & o & o & o & c & c & o & o & o \\ o & o & o & o & o & o & o & o & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \end{bmatrix} \qquad (5.1)$$

A test will first be generated for line x_4 stuck-at-0. As in the D-algorithm, the letter D will indicate the value of a line that is 1 in a fault-free circuit, but has taken the value 0 due to the presence of a fault (D = 1/0).

On the right-hand side of Equation (5.1), replace x_4 by the value D. This substitution is labeled as Step 1 in Equation (5.2).

Column 4 indicates that line x_4 connects to line x_9 and the error propagation rules indicate that each of the other nonnull variables in row x_9 must take the value 1. These values ($x_2 = u_2 = 1$, $x_6 = 1$, and $x_9 = D'$) are entered in the right-hand vector and labeled Step 2.

The error signal has been propagated to line x_9. Column 9 indicates that line x_9 connects to primary output x_{11} and the error propagation rules indicate that the nonnull variables, x_{10}, in row 11, take the value 1. These values (x_{11} = D and $x_{10} = 1$) are entered in the right-

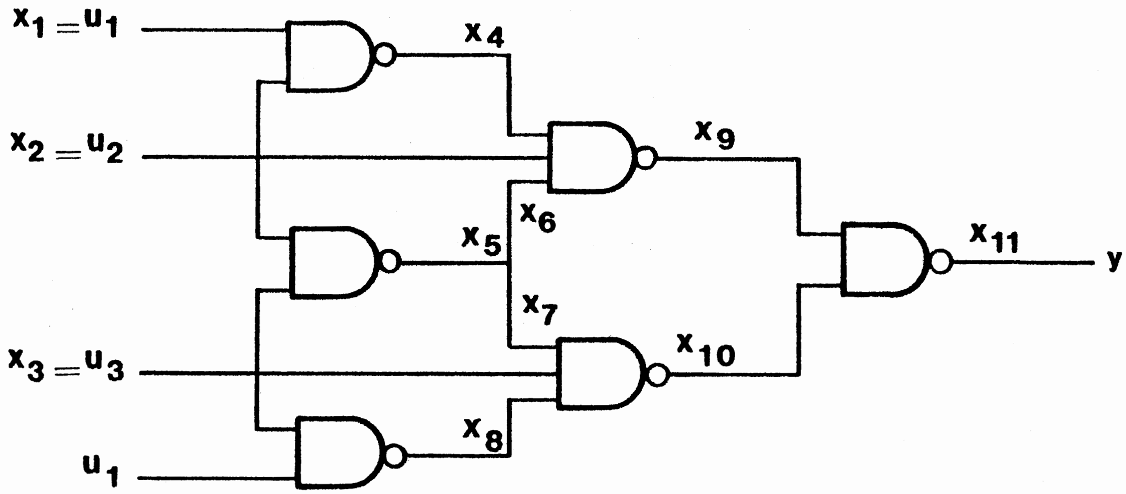


Figure 18. Labeled Combinational Circuit

hand vector and labeled Step 3. Due to the nature of the propagation procedure, the variables x_9 and x_{11} will not require justification and can be removed from the list of variables to be justified.

The justification procedure now begins. The signal $x_4 = 1$ (the fault-free value on the faulty signal line) is justified if $x_1 = 0$ or $x_2 = 0$. Since $x_2 = 1$ has been previously assigned, we choose $x_1 = 0$. This assignment is labeled Step 4.

To ensure consistency, the implications of this assignment are made immediately. Column 1 indicates that x_1 connects to x_4 and x_8. Since $x_1 = 0$ and the entry in row 8, column 1 is c, then x_8 must equal 1. This implication is entered as Step 5. The implication assignment of $x_8 = 1$ implies that x_8 has been justified (since $x_1 = 0$); thus x_8 can be removed from the list of variables which need to be justified.

The justification procedure is continued by justifying $x_6 = 1$. The justification rules indicate that $x_5 = 1$ (Step 6).

The implication of this assignment is made by investigating column 5 which leads to $x_7 = 1$ (Step 7). x_7 can now be removed from the list of variables requiring justification. Only x_{10} and x_5 remain to be justified.

The value $x_{10} = 1$ is now justified. The rules indicate that either $x_3 = 0$, $x_7 = 0$, or $x_8 = 0$. Since $x_7 = x_8 = 1$ has already been assigned, the zero value is given to x_3 (Step 8). Implications of the assignment $x_3 = 0$ indicate that $x_5 = 1$, $x_8 = 1$, and $x_{10} = 1$. It is noted that $x_8 = x_{10} = 1$ is consistent with previous assignments, and x_5 is removed from the list requiring justification.

The test derivation is complete. The test $T = (u_1, u_2, u_3) = (x_1, x_2, x_3) = (0, 1, 0)$ is a test for $x_4 = D$ (i.e., x_4 stuck-at-0). In addition, it will detect the faults $x_9 = D'$ and $x_{11} = D$.

$$
\begin{array}{c}
\text{Column No.} = \;1\;2\;3\;4\;5\;6\;7\;8\;9\;10\;11\;\;T
\end{array}
$$

$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \end{bmatrix}
=
\begin{bmatrix}
i & o & o & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o & o & o \\
c & c & o & o & o & o & o & o & o & o & o \\
o & c & c & o & o & o & o & o & o & o & o \\
o & o & o & o & i & o & o & o & o & o & o \\
o & o & o & o & i & o & o & o & o & o & o \\
c & o & c & o & o & o & o & o & o & o & o \\
o & c & o & c & o & c & o & o & o & o & o \\
o & o & c & o & o & o & c & c & o & o & o \\
o & o & o & o & o & o & o & o & c & c & o
\end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \\ D \\ 1 \\ 1 \\ 1 \\ 1 \\ D' \\ 1 \\ D \end{bmatrix}
\begin{array}{l} \text{Step 4} \\ \text{Step 2} \\ \text{Step 8} \\ \text{Step 1} \\ \text{Step 6} \\ \text{Step 2} \\ \text{Step 7} \\ \text{Step 5} \\ \text{Step 2} \\ \text{Step 3} \\ \text{Step 3} \end{array}
\qquad (5.2)
$$

In the above example, we were "lucky" because in instances where choices had to be made, the values could be uniquely determined (Steps 4 and 8). The next example illustrates the procedure when choices must be made.

Figure 19 is a small combinational circuit whose matrix representation is given by Equation (5.3) (see page 63).

Derive a test for x_9 stuck-at-0 (Step 1 in Equation [5.4]). Propagate the fault $x_9 = D$. Column 9 indicates that x_9 connects to x_{12} and the fault is propagated if $x_8 = 1$ (Step 2). Propagate $x_{12} = D'$. Column 12 indicates that x_{12} connects to the primary output variable x_{13} and the fault is propagated if $x_{10} = 1$ and $x_{11} = 0$ (Step 3).

The justification procedure begins by justifying $x_9 = 1$. This is true if $x_1 = 0$ or $x_6 = 0$. Which should we choose? Column 1 indicates that x_1 fans out to line x_{10} as well as x_9. Check to see if $x_1 = 0$ would interfere with the value already assigned to x_{10}. It does not; in fact, it will justify $x_{10} = 1$. Therefore, the assignment $x_1 = 0$ is

$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{bmatrix}
=
\begin{bmatrix}
i & o & o & o & o & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o & o & o & o & o \\
o & o & o & i & o & o & o & o & o & o & o & o & o \\
o & o & o & o & i & o & o & o & o & o & o & o & o \\
o & o & o & o & o & i & o & o & o & o & o & o & o \\
o & o & o & o & o & o & i & o & o & o & o & o & o \\
o & c & o & o & o & o & c & o & o & o & o & o & o \\
c & o & o & o & o & c & o & o & o & o & o & o & o \\
c & o & c & o & o & c & o & o & o & o & o & o & o \\
o & o & c & c & c & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & c & c & o & o & o & o \\
o & o & o & o & o & o & o & o & o & c & i & c & o
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{bmatrix}
\qquad (5.3)
$$

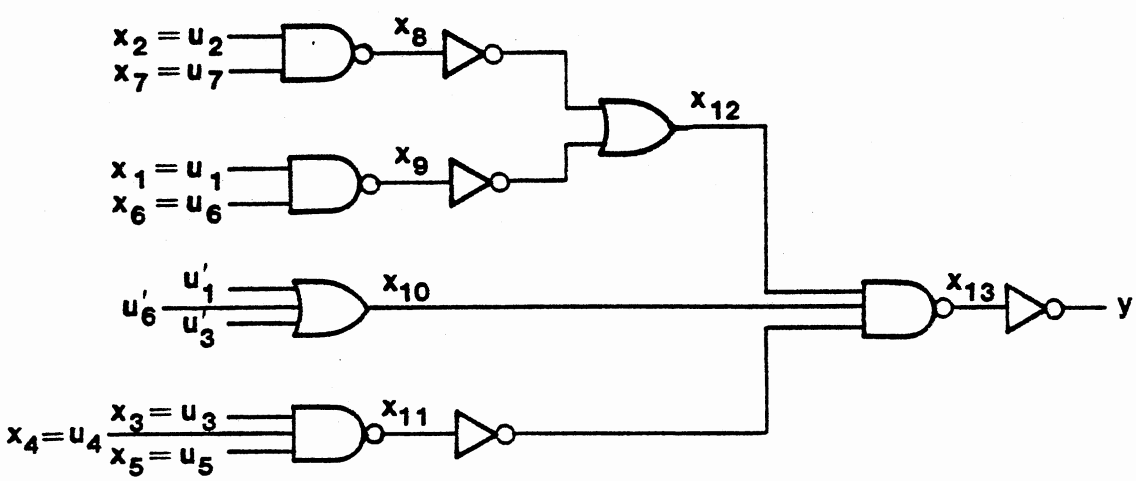


Figure 19. Combinational Circuit

made. It turns out in this case that the assignment $x_6 = 0$ is also valid because column 6 indicates that x_6 also connects to x_{10} as well as x_9, and the same argument would hold for assigning it the value 0. Once a value that "works" is found, however, the search is abandoned. Therefore, we assign the value $x_1 = 0$ (Step 4) with no fear of conflicts later.

Next justify $x_8 = 1$. A choice between $x_2 = 0$ or $x_7 = 0$ exists. Column 2 indicates that x_2 only fans out to x_8 and its assignment will not affect any other variable. Hence, we make the assignment $x_2 = 0$ (Step 5).

Next justify $x_{11} = 0$. We have no choice here. We must assign $x_3 = x_4 = x_5 = 1$ (Step 6).

Since no other unjustified values remain, the test is complete. The unassigned inputs, u_6 and u_7, are treated as "don't care" variables. The test $(u_1, u_2, u_3, u_4, u_5, u_6, u_7) = (0, 0, 1, 1, 1, x, x)$ detects the faults x_9 and x_{13} stuck-at-0 and x_{12} stuck-at-1.

$$
\begin{array}{cccc}
 & \text{Column No.} = & 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13 & \text{T} \\
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} & = &
\begin{bmatrix}
i & o & o & o & o & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o & o & o & o & o \\
o & o & o & i & o & o & o & o & o & o & o & o & o \\
o & o & o & o & i & o & o & o & o & o & o & o & o \\
o & o & o & o & o & i & o & o & o & o & o & o & o \\
o & o & o & o & o & o & i & o & o & o & o & o & o \\
o & c & o & o & o & o & c & o & o & o & o & o & o \\
c & o & o & o & o & c & o & o & o & o & o & o & o \\
c & o & c & o & o & c & o & o & o & o & o & o & o \\
o & o & c & c & c & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & c & c & o & o & o & o \\
o & o & o & o & o & o & o & o & o & c & i & c & o
\end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ \\ \\ 1 \\ D \\ 1 \\ 0 \\ D' \\ D \end{bmatrix}
\begin{array}{l} \text{Step 4} \\ \text{Step 5} \\ \text{Step 6} \\ \text{Step 6} \\ \text{Step 6} \\ \\ \\ \text{Step 2} \\ \text{Step 1} \\ \text{Step 3} \\ \text{Step 3} \\ \text{Step 2} \\ \text{Step 3} \end{array}
\end{array}
\qquad (5.4)
$$

Fanout paths that reconverge are referred to as reconvengent fanout paths. Schneider (1967) showed, through example, that some faults on a reconvergent fanout path cannot be detected by single path sensitization but can be detected through multiple path sensitization. Armstrong (1966) showed that multiple path sensitization can, in some cases, prohibit the effect of a fault from propagating beyond the point of reconvergence. One case, in particular, where this situation can occur is when some reconverging fanout paths between a specified node and a specified node of reconvergence have different inversion parity. Armstrong defines the inversion parity of a reconvergent fanout path to be the number of inversions, modulo 2, along the path between the specified fanout node and the specified node of reconvergence.

This presents a dilemma. If only single path sensitization is performed, some faults may remain undetected and a trial-and-error exhaustion of all single paths must be done before discovering the need for multiple path sensitization. If a procedure involving only multiple path sensitization is followed, some faults may be untestable because of reconvergent complemented fanout.

Several approaches have been taken with regard to this problem by path sensitization algorithms. The D-algorithm is capable of either single or multiple path sensitization. However, multiple path sensitization is extremely time-consuming when done by the D-algorithm. Breuer (1981) asserts that the need for multiple path sensitization does not occur often enough to warrant the time expenditure when generating tests for large systems. D-LASAR can detect some, but not all, faults for which multiple path sensitization is required. In most cases, no attempt is made to detect those faults not found by the standard algorithm.

The Boolean difference algorithm, in its most general form, can detect faults requiring multiple path sensitization and is not hindered by reconvergent complemented fanout. This is due to the fact that the general Boolean difference formula covers each single path (first-order terms) and all possible multiple paths (higher-order terms). To speed up the algorithm, the higher-order terms may be omitted.

In general, the approach to be adopted in this thesis is to follow the single path procedure in order to avoid the problem associated with reconvergent complemented fanout. One advantage lost due to the single path assumption is that more faults can be detected at once if multiple paths are sensitized. For reference, however, the following example illustrates how multiple path sensitization can be accomplished using the matrix formulation.

Figure 20 is Schneider's example (1967). The fault x_6 stuck-at-1 requires multiple path sensitization. The matrix formulation of Schneider's circuit is given in Equation (5.5).

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} i & o & o & o & o & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o & o & o & o & o \\ o & o & i & o & o & o & o & o & o & o & o & o \\ o & o & o & i & o & o & o & o & o & o & o & o \\ i & o & i & o & o & o & o & o & o & o & o & o \\ o & i & i & o & o & o & o & o & o & o & o & o \\ o & i & o & i & o & o & o & o & o & o & o & o \\ o & i & o & o & c & o & o & o & o & o & o & o \\ i & o & o & o & o & c & o & o & o & o & o & o \\ o & o & o & i & o & c & o & o & o & o & o & o \\ o & o & i & o & o & o & c & o & o & o & o & o \\ o & o & o & o & o & o & o & c & c & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} \tag{5.5}$$

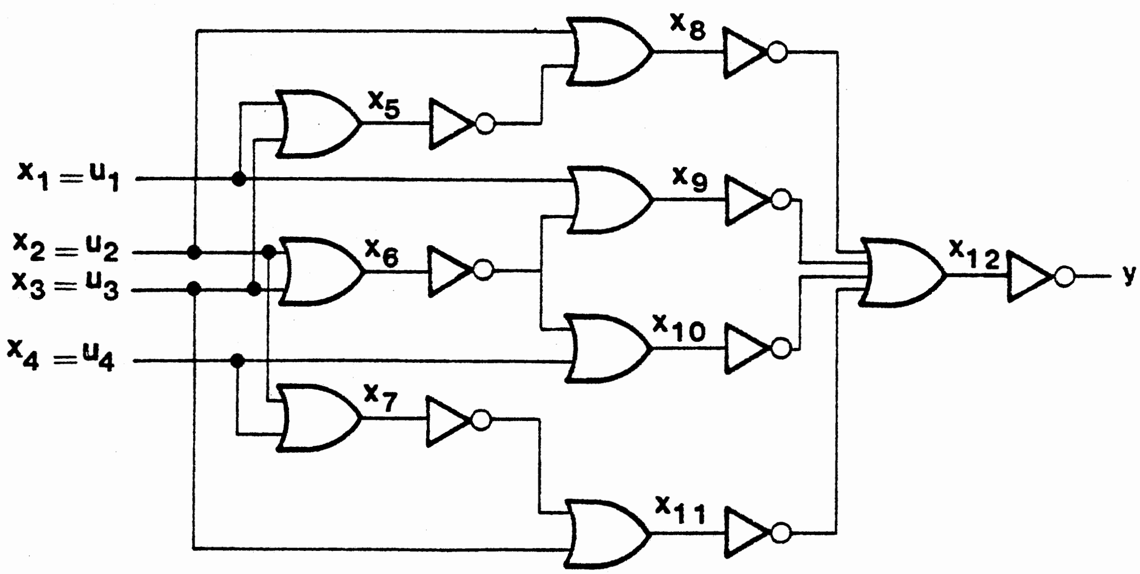


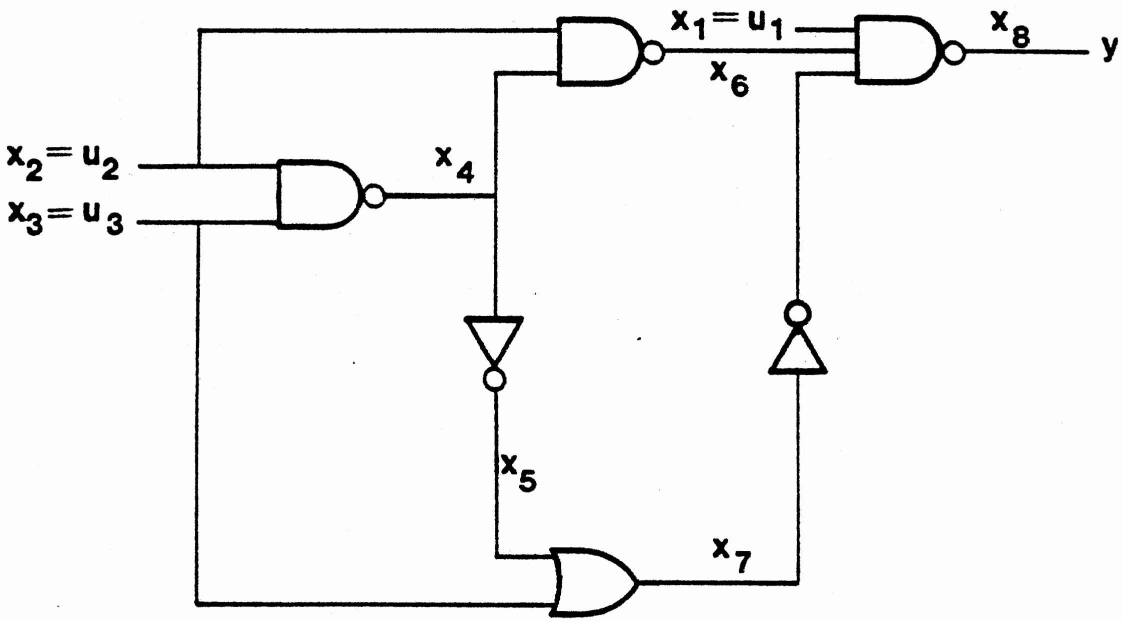
Figure 20. Schneider's Example (Schneider, 1967)

The column on the right of Equation (5.6) shows the steps involved in double path sensitization.

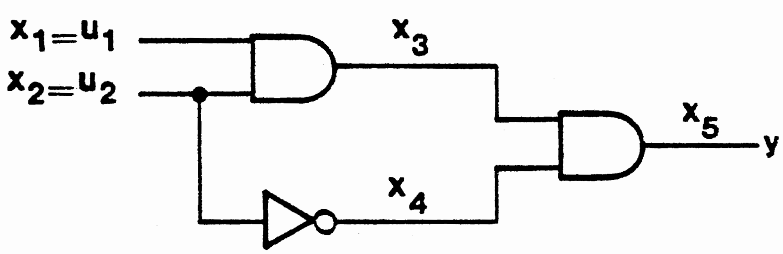
$$
\begin{array}{ccccc}
 & & \text{Column No.} = \;\; 1\;\; 2\;\; 3\;\; 4\;\; 5\;\; 6\;\; 7\;\; 8\;\; 9\;\; 10\;\; 11\;\; 12 & T & \text{Step} \\
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix}
& = &
\begin{bmatrix}
i & o & o & o & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o & o & o & o \\
o & o & o & i & o & o & o & o & o & o & o & o \\
i & o & i & o & o & o & o & o & o & o & o & o \\
o & i & i & o & o & o & o & o & o & o & o & o \\
o & i & o & i & o & o & o & o & o & o & o & o \\
o & i & o & o & c & o & o & o & o & o & o & o \\
i & o & o & o & o & c & o & o & o & o & o & o \\
o & o & o & i & o & c & o & o & o & o & o & o \\
o & o & i & o & o & o & c & o & o & o & o & o \\
o & o & o & o & o & o & o & c & c & c & c & o
\end{bmatrix}
&
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ D' \\ 0 \\ 1 \\ D \\ D \\ 1 \\ D' \end{bmatrix}
&
\begin{array}{c} (2) \\ (4) \\ (4) \\ (2) \\ (5) \\ (1) \\ (6) \\ (3) \\ (2) \\ (2) \\ (3) \\ (3) \end{array}
\end{array}
\qquad (5.6)
$$

Because of the way a circuit is designed, some faults may not be testable. This is the case when redundancy is built into the circuit. The single fault model is not appropriate in this case because the circuit redundancy masks single faults.

Figure 21(a) illustrates a circuit containing an undetectable fault. The value 0 on line x_4 can never be observed at a primary output. This means that a test cannot be generated for the fault x_5 stuck-at-1. Some faults may not be detectable because it is not possible to justify the line to a particular value. Figure 21(b) illustrates this situation in a very simple circuit. The line x_5 can never be set to the value 1 and hence can never be tested for the stuck-at-0 condition. Both circuits are nonminimal constructs, and each possesses reconvergent complemented fanout.



(a) Circuit With Unobservable 0 on Line x_4



(b) Circuit With Unjustifiable 1 on Line x_5

Figure 21. Circuits With Undetectable Faults

A test generation algorithm can waste a lot of time trying to generate a test for an undetectable fault. The test algorithm needs to be able to determine, within a reasonable length of time, whether a fault is undetectable. The path sensitization approach of Roth (1966), implemented either in the D-algorithm or D-LASAR, can determine when a fault is undetectable. When the matrix model is used in conjunction with path sensitization, undetectable faults can be similarly identified.

The steps involved in determining that the fault x_4 = D' in Figure 20(a) is not detectable are shown in Equation (5.7). It is determined in Step 5 that the value D' on line x_4 cannot be justified in a consistent manner with the values specified in the propagation phase. The fault-free value $x_4 = 0$ can never be achieved as long as $x_3 = 0$. If one attempts to propagate the error x_4 = D' along the path through x_6 to x_8, the value of x_7 cannot be justified. Double path sensitization also fails because x_7 cannot be justified consistent with the assignments made during the propagation phase and during the justification of the fault-free value $x_4 = 0$.

$$\begin{array}{ccccc}
 & & \text{Column No.} = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 & \text{T} & \text{Step} \\
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} & = & \begin{bmatrix} i & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o \\ o & o & i & o & o & o & o & o \\ o & c & c & o & o & o & o & o \\ o & o & o & c & o & o & o & o \\ o & c & o & c & o & o & o & o \\ o & o & i & o & i & o & o & o \\ c & o & o & o & o & c & i & o \end{bmatrix} & \begin{bmatrix} 1 \\ \\ 0 \\ D' \\ D \\ 1 \\ D \\ D \end{bmatrix} & \begin{matrix} (4) \\ \\ (3) \\ (1) \\ (2) \\ (4) \\ (3) \\ (4) \end{matrix}
\end{array} \qquad (5.7)$$

The iterative combinational model of sequential circuits formulated in matrix form provides a convenient tool for test generation using the path sensitization approach. To illustrate the self-initialization procedure, a test will be generated for $x_4(t)$ stuck-at-0 in the circuit pictured in Figure 22.

The matrix formulation for the clocked S-R flip-flop is given by Equation (5.8).

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \end{bmatrix} = \begin{bmatrix} i & o & o & o & o & o & o & o & o \\ o & i & o & o & o & o & o & o & o \\ o & o & i & o & o & o & o & o & o \\ c & o & c & o & o & o & o & o & o \\ o & c & c & o & o & o & o & o & o \\ o & o & o & c & o & o & o & o & c \\ o & o & o & o & c & o & o & c & o \end{bmatrix} \begin{bmatrix} S(t) \\ R(t) \\ CLK(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \\ x_6(t-1) \\ x_7(t-1) \end{bmatrix} \tag{5.8}$$

Test generation begins by assigning $x_4(t) = D$. See Equation (5.9). Inspection of column 4 indicates that the fault can be propagated to primary output $x_6(t)$ if $x_7(t-1) = 1$ (Step 2). To justify a D value on line $x_4(t)$ requires either $x_1(t) = 0$ or $x_3(t) = 0$. Since $x_1(t)$ does not fanout, it is assigned the zero value (Step 3).

The value $x_7(t-1)$ is shifted into the x_7 position of the time t-1 vector (Step 4) and the D value on $x_4(t-1)$ justified. Justification of $x_7(t-1) = 1$ is performed in Step 5 by assigning $x_5(t-1) = 0$. The value on $x_5(t-1)$ is justified in Step 6 by assigning $x_2(t-1) = x_3(t-1) = 1$. The test sequence is complete. It is (0, 1, 1) followed by (0, x, x).

The flow diagram in Figure 23 indicates the general procedure for combinational circuit test generation using the matrix model. The vector T in the flowchart corresponds to the vector on the right-hand side of the matrix equation $X = M[U\ X]^T$. It is initially in an unknown state

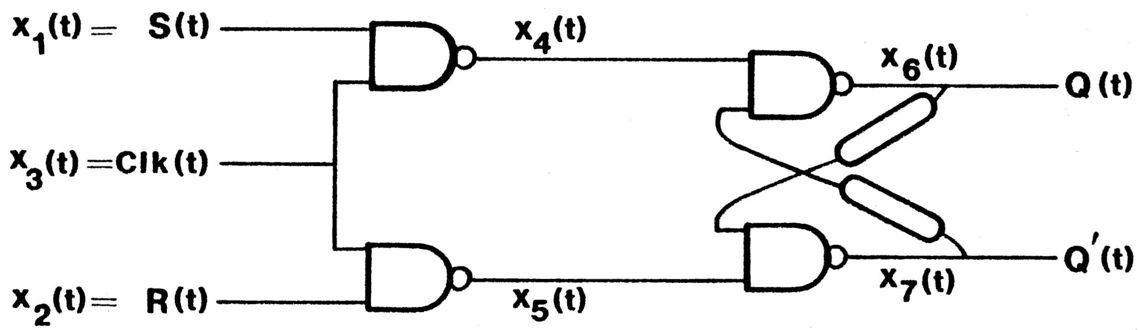


Figure 22. Gate Level Representation of a Clocked S-R Flip Flop

Column No. = 1 2 3 4 5 6 7 -6 -7 T(t) Step T(t-1) Step

$$
\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \end{bmatrix}
=
\begin{bmatrix}
i & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o \\
c & o & c & o & o & o & o & o & o \\
o & c & c & o & o & o & o & o & o \\
o & o & o & c & o & o & o & o & c \\
o & o & o & o & c & o & o & c & o
\end{bmatrix}
\left[\begin{array}{c} 0 \\ \\ \\ D \\ \\ D' \\ \\ \hline 1 \end{array}\right]
\begin{array}{c} (3) \\ \\ \\ (1) \\ \\ (2) \\ \\ (2) \end{array}
\quad
\left[\begin{array}{c} 0 \\ 1 \\ 1 \\ D \\ 0 \\ \\ 1 \\ \hline \\ \end{array}\right]
\begin{array}{c} (4) \\ (6) \\ (6) \\ (1) \\ (5) \\ \\ (4) \\ \\ \end{array}
\qquad (5.9)
$$

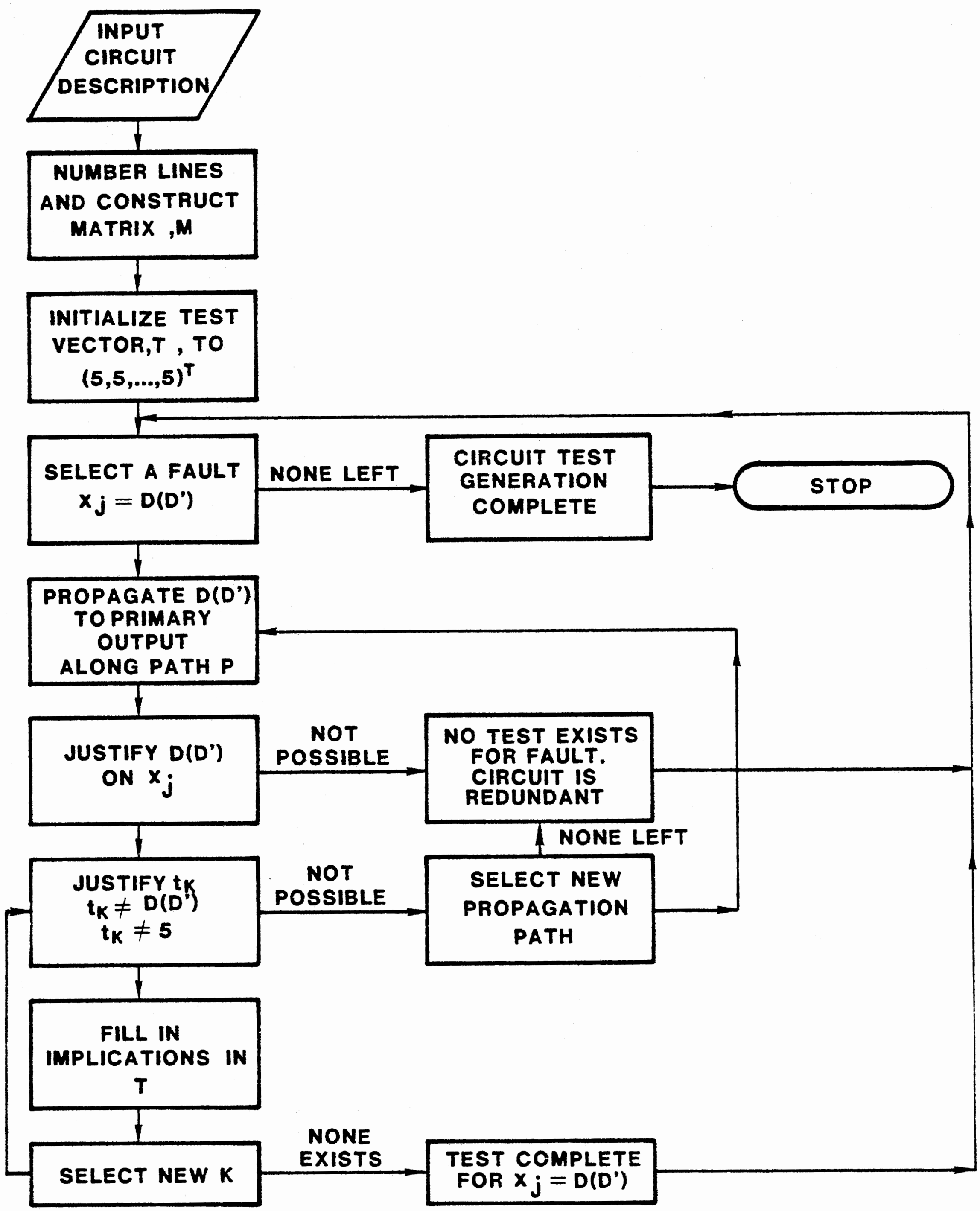


Figure 23. Flow Diagram for Combinational Circuit Test Generation Using Single Path Sensitization and the Acken Matrix Model

with each entry denoted as $t_k = 5$. When justifying t_k ($t_k \neq D(D')$ or $t_k \neq 5$), choices may exist and should be handled in the manner described for the circuit of Figure 19.

Boolean Difference Using the Matrix Model

The Boolean difference approach to ATG is essentially an equation manipulation procedure which can, in its most general form, generate all possible tests for stuck-at faults. An algorithm which uses the Boolean difference as a means to generate complete test sets for combinational logic circuits has been set forth by Yau and Tang (1971). This algorithm will be used to illustrate the ways in which the matrix formulation can be used in conjunction with Boolean difference.

Stated briefly, the steps of the algorithm include:

1. Number all lines and write the Boolean equation for each line x_j and its complement x_j' in terms of the primary input variables. List them in Table A.

2. Find the Boolean difference of the logic function f, realized by the logic circuit, with respect to each fanout line using the definition of Boolean difference, i.e., compute $df/dx_j = f(x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n) \oplus f(x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_n)$, where j ranges over all fanout lines.

3. Find the Boolean differences df/dx_j, where j starts from the largest line number (a primary output) to the smallest line number that is not a fanout line. List them in Table B.

4. From Tables A and B, express $x_j(df/dx_j)$ and $x_j'(df/dx_j)$, $j = 1, 2, \ldots, n$, in terms of the primary input variables. The set of terms

of $x_j(df/dx_j)$ and $x_j'(df/dx_j)$ represent the complete test set of the single permanent stuck-at faults on line j.

The matrix formulation is particularly convenient in Steps 1 and 3. Step 1 in the Yau and Tang algorithm calls for numbering the lines and writing the logic equation for every line in terms of primary inputs. The following algorithm can be used to generate these equations using the matrix model.

First, partition the matrix M as in Equation (5.10) and remove the data steps. See Equation (5.11).

$$\begin{bmatrix} X_n \\ X_r \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} U \\ X_r \end{bmatrix} \tag{5.10}$$

$$X_r = \begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} U \\ X_r \end{bmatrix} \tag{5.11}$$

When expanded, Equation (5.11) can be written

$$X_r = A_1 U + A_2 X_r \ . \tag{5.12}$$

The A_2 matrix is lower triangular because of the line numbering convention. The entries in the matrices A_1 and A_2 are Boolean functions of one variable from the set $C = \{o, i, c, u\}$.

The algorithm to generate the logic equations for the lines x_j in the logic circuit, in terms of the primary inputs, proceeds as follows:

Step 1. Write the matrix equation leaving out the data steps as in Equation (5.12). Expand the right-hand side into a vector.

Step 2. Operate on the vector generated in Step 1, by A_2.

Step 3. Add the vector $A_1 U$.

Step 4. Check to see if any entry in the vector generated by Step 3 contains any x-variables.

Yes--go to Step 2.

No--Stop.

This algorithm will be illustrated for the example of Figure 24 taken from Yau and Tang. The complete matrix equation is given in Equation (5.13).

$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
=
\begin{bmatrix}
i & o & o & o & o & o & o & o & o & o & o & o & o & o & o & o \\
o & i & o & o & o & o & o & o & o & o & o & o & o & o & o & o \\
o & o & i & o & o & o & o & o & o & o & o & o & o & o & o & o \\
o & o & o & i & o & o & o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & i & o & o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & i & o & o & o & o & o & o & o & o & o & o \\
c & c & o & o & o & o & o & o & o & o & o & o & o & o & o & o \\
o & o & c & c & o & o & o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & c & c & o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & c & o & o & i & o & o & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o & c & o & o & o & o & o \\
o & o & o & o & o & o & o & o & c & o & o & i & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o & o & o & c & c & c & o
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
\tag{5.13}
$$

The reduced equation is given by Equation (5.14).

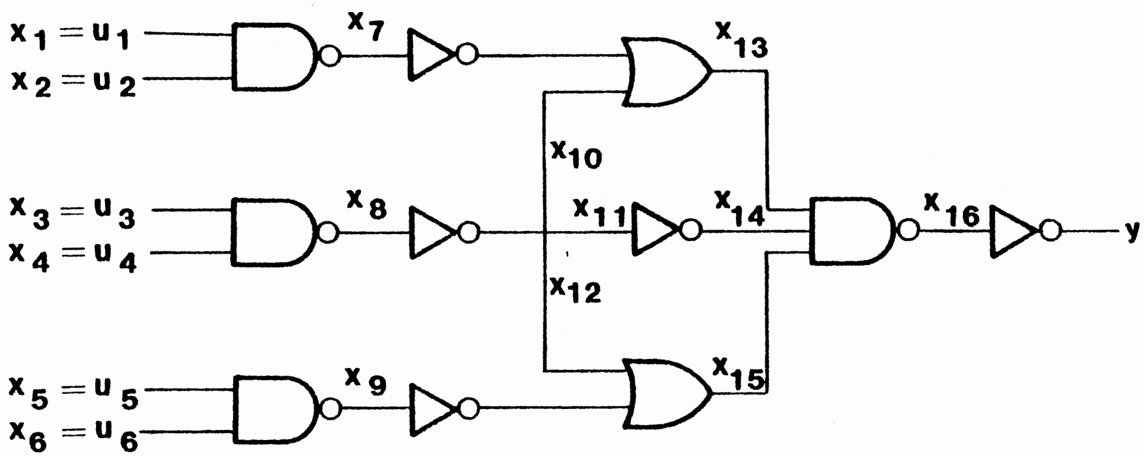


Figure 24. Example Circuit for Illustrating Boolean Difference (Yau and Tang, 1971)

$$
\begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
=
\begin{bmatrix}
c & c & o & o & o & o \\
o & o & c & c & o & o \\
o & o & o & o & c & c \\
o & o & o & o & o & o \\
o & o & o & o & o & o \\
o & o & o & o & o & o \\
o & o & o & o & o & o \\
o & o & o & o & o & o \\
o & o & o & o & o & o \\
o & o & o & o & o & o
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}
+
\begin{bmatrix}
o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
c & o & o & i & o & o & o & o & o & o \\
o & o & o & o & c & o & o & o & o & o \\
o & o & c & o & o & i & o & o & o & o \\
o & o & o & o & o & o & c & c & c & o
\end{bmatrix}
\begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
\tag{5.14}
$$

Computing the right-hand side of Equation (5.14) and writing the result in vector notation, we obtain

$$
X_r =
\begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix}
=
\begin{bmatrix}
cu_1 + cu_2 \\
cu_3 + cu_4 \\
cu_5 + cu_6 \\
cx_8 \\
cx_8 \\
cx_8 \\
cx_7 + x_{10} \\
cx_{11} \\
cx_9 + x_{12} \\
cx_{13} + cx_{14} + cx_{15}
\end{bmatrix}
=
\begin{bmatrix}
u_1' + u_2' \\
u_3' + u_4' \\
u_5' + u_6' \\
x_8' \\
x_8' \\
x_8' \\
x_7' + x_{10} \\
x_{11}' \\
x_9' + x_{12} \\
x_{13}' + x_{14}' + x_{15}'
\end{bmatrix}
\tag{5.15}
$$

Operating by A_2 on the left of X_r, we obtain

$$
\begin{bmatrix}
o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o \\
o & o & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
o & c & o & o & o & o & o & o & o & o \\
c & o & o & i & o & o & o & o & o & o \\
o & o & o & o & c & o & o & o & o & o \\
o & o & c & o & o & i & o & o & o & o \\
o & o & o & o & o & o & c & c & c & o
\end{bmatrix}
\begin{bmatrix}
u_1' + u_2' \\
u_3' + u_4' \\
u_5' + u_6' \\
x_8' \\
x_8' \\
x_8' \\
x_7' + x_{10} \\
x_{11}' \\
x_9' + x_{12} \\
x_{13}' + x_{14}' + x_{15}'
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
0 \\
u_3u_4 \\
u_3u_4 \\
u_3u_4 \\
u_1u_2 + x_8' \\
x_8 \\
u_5u_6 + x_8' \\
x_7x_{10}' + x_{11} + x_9x_{12}'
\end{bmatrix}
\tag{5.16}
$$

Add A_1U.

$$
\begin{bmatrix}
u_1' + u_2' \\
u_3' + u_4' \\
u_5' + u_6' \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
+
\begin{bmatrix}
0 \\
0 \\
0 \\
u_3u_4 \\
u_3u_4 \\
u_3u_4 \\
u_1u_2 + x_8' \\
x_8 \\
u_5u_6 + x_8' \\
x_7x_{10}' + x_{11} + x_9x_{12}'
\end{bmatrix}
=
\begin{bmatrix}
u_1' + u_2' \\
u_3' + u_4' \\
u_5' + u_6' \\
u_3u_4 \\
u_3u_4 \\
u_3u_4 \\
u_1u_2 + x_8' \\
x_8 \\
u_5u_6 + x_8' \\
x_7x_{10}' + x_{11} + x_{12}'
\end{bmatrix}
\tag{5.17}
$$

Since x_j appears on the right-hand side of Equation (5.17), another iteration is necessary. At the end of Step 3 in the third iteration, we obtain the vector

$$\begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix} = \begin{bmatrix} u_1' + u_2' \\ u_3' + u_4' \\ u_5' + u_6' \\ u_3u_4 \\ u_3u_4 \\ u_3u_4 \\ u_1u_2 + u_3u_4 \\ u_3' + u_4' \\ u_5u_6 + u_3u_4 \\ (u_1' + u_2')(u_3' + u_4') + u_3u_4 + (u_5' + u_6')(u_3' + u_4') \end{bmatrix} \tag{5.18}$$

which contains only primary input variables. This vector contains the Boolean equation for each internal line in terms of primary inputs.

In Step 3 of the Yau and Tang algorithm, the Boolean differences df/dx_j must be computed. The index j progresses from the largest line number to the smallest line number that is not a fanout line. These differences are easily calculated using the matrix model and the following lemmas.

Lemma 1 (Sellers, Hsiao, and Bearnson, 1968):

$$\frac{df}{dx_i} = \frac{df}{dx_i'}$$

Lemma 2 (Chaing, Reed, and Banes, 1972): If

$$f(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \ldots + x_n ,$$

then

$$\frac{df}{dx_i} = x_1' \cdot x_2' \ldots x_{i-1}' \cdot x_{i+1}' \ldots x_n' .$$

If

$$f(x_1, x_2, \ldots, x_n) = x_1 x_2 \ldots x_n ,$$

then

$$\frac{df}{dx_i} = x_1 \cdot x_2 \ldots x_{i-1} \cdot x_{i+1} \ldots x_n .$$

Since the matrix model expresses each line in terms of the sum of "preceding" lines, Lemma 2 can be directly applied. For reference, the matrix equation for the circuit of Figure 24 is repeated. The data steps are again removed.

$$\begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix} = \begin{bmatrix} c & c & o & o & o & o & o & o & o & o & o & o & o & o & o & o \\ o & o & c & c & o & o & o & o & o & o & o & o & o & o & o & o \\ o & o & o & o & o & c & c & o & o & o & o & o & o & o & o & o \\ o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\ o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\ o & o & o & o & o & o & o & c & o & o & o & o & o & o & o & o \\ o & o & o & o & o & o & c & o & o & i & o & o & o & o & o & o \\ o & o & o & o & o & o & o & o & o & o & c & o & o & o & o & o \\ o & o & o & o & o & o & o & o & c & o & o & i & o & o & o & o \\ o & o & o & o & o & o & o & o & o & o & o & o & c & c & c & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix} \qquad (5.13a)$$

Beginning with $j = 16$, we see from the matrix that $x_{16} = x'_{13} + x'_{14} + x'_{15}$. It follows from Lemmas 1 and 2 that

$$\frac{df}{dx_{16}} = \frac{dx'_{16}}{dx_{16}} = 1$$

$$\frac{df}{dx_{15}} = \frac{dx'_{16}}{dx_{15}} = x_{13}\ x_{14}$$

$$\frac{df}{dx_{14}} = \frac{dx'_{16}}{dx_{14}} = x_{13}\ x_{15}$$

$$\frac{df}{dx_{13}} = \frac{dx'_{16}}{dx_{13}} = x_{14}\ x_{15}$$

By inspection of the matrix in Equation (13a), we have

$$\frac{dx_{15}}{dx_{12}} = x_9$$

$$\frac{dx_{15}}{dx_9} = x'_{12}$$

There is no need to compute these differences and store them in tabular form as the Yau and Tang algorithm suggest since they can be computed by inspection of the matrix.

Summary

In this chapter, the matrix model for digital systems forms the framework through which a path sensitization algorithm is carried out. Using this technique and structure, tests for single permanent stuck-at faults are generated for both combinational and sequential circuits. The model contains both behavioral and topological information which enables error propagation and line value justification to be performed in a concise methodical manner.

An algorithm based upon the matrix model is introduced for deriving the Boolean equation of each internal line in terms of primary inputs. Boolean differences of the function with respect to each line can be computed directly by inspection of the defining matrix.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

A mathematical model of digital systems formulated for use by automatic test generation algorithms has been described in this thesis. The general formulation and capabilities of the model are exhibited through numerous examples.

Gate level automatic test generation algorithms can benefit from two types of circuit information, behavioral and topological. The matrix model relates the behavioral information through the data and computation steps of a k-step chain. The steps are written in a linear-looking form by using the basis functions OR and NOT. Gate level circuit diagrams are transformed into this basis by using only NAND, OR, and NOT gates. The resulting matrix equation, in addition to containing behavioral information, contains the same topological information held in the adjacency matrix of the logic gate-level circuit diagram. The correlation of the topological information to the gate level logic diagram is valuable in the path sensitization approach because it contains information about the various paths from inputs to outputs. Fanin (fanout) information is also available for each gate.

The number of nonnull elements in the defining matrix of the model is equal to the number of primary inputs plus the sum of all fanin (fanout) connections in the logic circuit description. The linear storage complexity of the model indicates the probable ease of extension to the modeling of very large circuits.

Examples were given which illustrate the use of the matrix model as a structure in which the path sensitization approach to automatic test generation and the Boolean difference approach can be carried out.

The concise matrix formulation of digital systems will offer considerable potential for use in a variety of digital systems analysis problems. Several topics for further research will be discussed in the next section.

Recommendations for Further Research

Within the analytical framework established by this thesis, several additional areas of research arise for the application of the matrix model to digital systems analysis. The problems are, for the most part, associated with the efficient utilization of the topological information contained in the defining matrix to address problems associated with automatic test generation.

The ATG algorithms based upon the approaches of Roth (1966) and Sellers, Hsiao, and Bearnson (1968) utilize the topological information of the matrix model implicitly in test generation. Explicit utilization of this information to determine in advance which faults will require multiple path sensitization or to locate paths which contain reconvergent complemented fanout could increase the computation time efficiency of the algorithms in generating tests for all possible stuck-type faults.

One possible approach to identifying paths through a network having particular properties is to determine the pattern of a submatrix with the desired property. Next, introduce various row and column interchange operations on the defining matrix which will result in forming the pattern in a submatrix when such a pattern exists.

Consider a circuit which realizes the function f_α in the presence of a given fault α, and the function f in the fault-free instance. If $f_\alpha \oplus f = 0$, then the fault α is said to be undetectable and the circuit is said to be redundant with respect to the fault α. When a circuit is known to be redundant with respect to a stuck-type fault, certain circuit simplification rules can be applied to condition the circuit for automatic test generation (see Breuer and Friedman, 1976). In general, there is no simple way to determine when redundancy in combinational circuits exist other than proving that no test exists for some fault or faults. The method of proof is to try to generate a test and ultimately show that none exists. A great amount of computation may be required for this method. It is desirable to determine the redundancy prior to test generation.

The circuit in Figure 25 illustrates an extremely simple redundant circuit. The matrix equations for this circuit are

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} o & o & o & o & o \\ o & o & o & o & o \\ c & c & o & o & o \\ c & c & o & o & o \\ o & o & i & i & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} i & o \\ o & i \\ o & o \\ o & o \\ o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$[y] = [o\ o\ o\ o\ i]\ X\ .$$

Note that rows 3 and 4 in the A-matrix are identical, indicating a redundancy. The redundancy can be eliminated from the circuit by deleting the row and column associated with x_3 (or x_4). The resulting matrix equations are

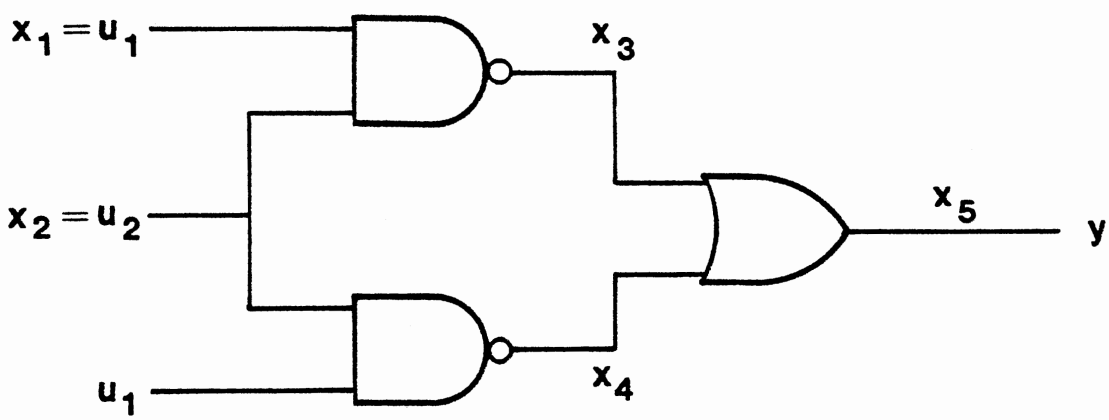


Figure 25. A Simple Redundant Circuit

$$\begin{bmatrix} x_1 \\ x_2 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} o & o & o & o \\ o & o & o & o \\ c & c & o & o \\ o & o & i & o \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} i & o \\ o & i \\ o & o \\ o & o \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$[y] = [o \ o \ o \ i] \ X \ .$$

Not all redundancies are this obvious, but this simple example seems to indicate that certain row and column operations could be defined which transform the A-matrix into one containing two or more identical rows (columns), thus indicating redundancies.

The third area in which the matrix model for digital systems has potential application is that of testability analysis. Testability analysis deals with the determination of measures which describe how easily the DUT can be tested. The measures can describe either the intrinsic or extrinsic testability of the device. Intrinsic testability measures consider the circuit behavior and topology alone and do not depend upon stimulus/response considerations. They are computed during the early design phase to provide an indication of how difficult it will be to generate test vectors either manually or automatically. Extrinsic testability measures are associated with a particular test. After the test has been generated, either manually or automatically, it is graded to determine the percent of faults detected and/or located. This percentage is the extrinsic testability measure and is generally computed by a fault simulator which generates statistics. Extrinsic testability measures can also be computed during the design phase.

Goldstein (1979) defined some intrinsic testability measures of controllability and observability. The controllability of a combinational node (e.g., AND, OR, NOR, NAND gates or primary input nodes) is a measure of the minimum number of combinational node assignments in the circuit required to justify a zero or a one value on the node. The observability measure of a combinational node is related to both the distance between the node and a primary output and the difficulty of propagating the logical value on the node to a primary output. Sequential observability/controllability measures are similarly defined.

As in ATG, these intrinsic testability measures depend, to some extent, upon knowledge of the behavior of individual gates in the circuit and upon the interconnection of those gates (circuit topology). Since the matrix model contains this information in a concise formulation, it can perhaps be used as a structure for testability analysis. The specifics of the utilization of the model would depend upon the particular testability analysis procedure.

One way, in particular, that the model could be utilized in computing the observability measure is in the computation of the distance between a node and a primary output. The entries in the matrix, A, for combinational circuits correspond to the entries in the transposed adjacency matrix of the associated logic diagram. Hence, an entry a_{ij} in the matrix $(A^T)^r$ (A^T multiplied by itself r-times) is related to the number of different paths of length r between the ith and jth nodes (Deo, 1974).

A SELECTED BIBLIOGRAPHY

Abramovici, Miron. "Fault Diagnosis in Digital Circuits Based on Effect-Cause Analysis." (Unpub. Ph.D. dissertation, University of Southern California Electronic Sciences Laboratory, 1980.)

Agarwal, V. K., and G. M. Masson. "Recursive Coverage Projection of Test Sets." IEEE Trans. on Computers, C-28, 11 (1979), 865-870.

Akers, S. B. "On the Theory of Boolean Functions." SIAM J. of Applied Mathematics, 7 (1959), 487-498.

Akers, S. B. "A Logic System for Fault Test Generation." IEEE Trans. on Computers, C-25, 6 (1976), 620-630.

Armstrong, D. B. "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets." IEEE Transactions on Computers (1966), 66-73.

Batni, Ramachendra P., and C. R. Kime. "A Module-Level Testing Approach for Combinational Networks." IEEE Trans. on Computers, C-25, 6 (1976), 594-604.

Bearnson, L. W. "Arithmetic Error Detection in Digital Computers." (Unpub. M.S. thesis, Syracuse University, 1965.)

Bearnson, L. W., and C. C. Carroll. "On the Design of Minimum Length Fault Tests for Combinational Circuits." IEEE Trans. on Computers, 20 (1971), 1353-1356.

Bennetts, R. G., D. C. Brittle, A. C. Prior, and J. L. Washington. "A Modular Approach to Test Sequence Generation for Large Digital Networks." Digital Processes, 1 (1975), 3-24.

Bennetts, R. G. "Algebraic Models for Clocked Flip-Flops." Electronic Engineering (1978), 60-62.

Bouricius, W. G., E. P. Hsieh, G. R. Ptuzolu, J. P. Roth, P. R. Schneider, and C. J. Tan. "Algorithms for Detection of Faults in Logic Circuits." IEEE Trans. on Computers, C-20, 11 (1977), 1258-1264.

Breuer, M. A. "A Random and an Algorithmic Technique for Fault Detection Test Generation for Sequential Circuits." IEEE Trans. on Computers (1971), 1364-1370.

Breuer, M. A. "Testing for Intermittent Faults in Digital Circuits." IEEE Trans. on Computers, C-22, 3 (1973), 241-246.

Breuer, Melvin A. "Modeling Circuits for Test Generation." Digest of Papers. International Symposium on Fault Tolerant Computing, 1974, 3/13-3/18.

Breuer, Melvin A. "The Effects of Races, Delays, and Delay Faults on Test Generation." IEEE Trans. on Computers, C-23, 10 (1974), 1078-1092.

Breuer, Melvin A., and R. L. Harrison. "Procedures for Eliminating Static and Dynamic Hazards in Test Generation." IEEE Trans. on Computers, C-23, 19 (1974), 1069-1078.

Breuer, M. A., and A. D. Friedman. Diagnosis and Reliable Design of Digital Systems. New York: Computer Science Press, Inc., 1976.

Breuer, M. A., and A. D. Friedman. "Functional Level Primitives in Test Generation." IEEE Trans. on Computers, C-29, 3 (1980), 223-235.

Breuer, M. A. "Test Generation." (Presentation at Aerospace Symposium on Testing and Verification of Large Scale Integrated Circuits, Los Angeles, California, June, 1981.)

Brittle, D. C. "Notation Describing the Fault-Related Behavior of Logic Modules." Electronics Letters, 10-7 (1974), 215-216.

Brittle, David Charles. "Test Sequence Generation for Large Sequential Networks." (Unpub. Ph.D. dissertation, University of Southampton, England, 1976.)

Carroll, B. D., H. G. Shah, and D. M. Jones. "An Examination of Algebraic Test Generation Methods for Multiple Faults." IEEE Trans. on Computers, 23 (1974), 743-745.

Chappell, Stephen Gilbert. "Automatic Test Generation for Asynchronous Digital Circuits." Microfilm copy. (Unpub. Ph.D. dissertation, Library, Northwestern University, 1973.)

Chiang, A. C., I. S. Reed, and A. V. Banes. "Path Sensitization, Partial Boolean Difference, and Automated Fault Diagnosis." IEEE Trans. on Computers, C-21 (1972), 189-194.

Chiang, A. C. L., and Rick McCaskill. "Two New Approaches Simplify Testing of Microprocessors." Electronics, 49-2 (1976), 100-105.

Clegg, F. W. "Use of SPOOFs for Faulty Logic Network Analysis." IEEE International Symposium on Fault Tolerant Computing (1972), 143-147.

Deo, M. Graph Theory With Applications to Engineering and Computer Science. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1974.

Eichelberger, E. B., and T. W. Williams. "A Logic Design Structure for LSI Testability." Proc. 14th DAC, 1977.

Eldred, Richard D. "Test Routines Based on Symbolic Logic Statements." JACM, 6-1 (1959), 33-36.

Friedman, A. D., and P. R. Menon. Fault Detection in Digital Circuits. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1971.

Fujiwara, H., and K. Kinoshita. "On the Computational Complexity of System Diagnosis." IEEE Trans. on Computers, C-27, 10 (1978), 881-885.

Galiay, J., Y. Crouzet, and M. Vergniault. "Physical Versus Logical Fault Models in MOS LSI Circuits, Impact on Their Testability." Digest of Papers. International Symposium on Fault Tolerant Computing, 1979, 195-202.

Goldstein, Lawrence H. Computational Complexity/Confidence Level Trade-offs in LSI Testing. Report No. SAND77-2068. Albuquerque, N.M.: Sandia National Laboratories, 1978.

Goldstein, L. H. "Controllability/Observability Analysis of Digital Circuits." IEEE Trans. Circuits & Systems, CAS-26, 9 (1979), 685-693.

Harper, L. H., W. N. Hsieh, and J. E. Savage. "A Class of Boolean Functions With Linear Combinational Complexity." Theor. Comp. Science, 1-2 (1975), 161-183.

Hayes, J. P. "The Fanout Structure of Switching Functions." J. Assoc. Comput. Mach., 22 (1975), 551-571.

Hayes, John P. "Path Complexity of Logic Networks." IEEE Trans. on Computers, C-27, 5 (1978), 459-462.

Hsiao, M. Y., F. F. Sellers, and D. K. Chia. "Fundamentals of Boolean Difference for Test Pattern Generation." Proc. 4th Annual Princeton Conference on Information Sciences and Systems, 1970, 50-54.

Hsiao, M. Y., and D. K. Chia. "Boolean Difference for Fault Detection in Asynchronous Sequential Machines." IEEE Trans. on Computers, C-21 (1971), 1356-1361.

Hsieh, E. P., R. A. Rasmussen, L. J. Vidunas, and W. T. Davis. "Delay Test Generation." Proc. 14th Design Automation Conference, 1977, 486-491.

Ibarra, O. H., and S. K. Sahni. "Polynomially Complete Fault Detection Problems." IEEE Trans. on Computers, C-24, 3 (1975), 242-249.

Kodandapani, K. L, and S. C. Seth. "On Combinational Networks With Restricted Fan-Out." IEEE Trans. on Computers, C-27, 4 (1978), 309-138.

Kovijanic, P. G. "Testability Analysis." Proc. IEEE 1979 Semiconductor Test Conference, 1979, 310-316.

Ku, Chia-Tai, and G. M. Masson. "The Boolean Difference and Multiple-Fault Analysis." IEEE Trans. on Computers, C-24, 7 (1975), 691-695.

Kuhl, J. G., and S. M. Reddy. "On the Detection of Terminal Stuck Faults." IEEE Trans. on Computers, C-27, 5 (1978), 467-469.

Latino, Carl D., and J. G. Bredeson. "The Cell Method--A New Approach to Multiple Stuck-at-Fault Test Generation." International Journal of Electronics, 46-5 (1979), 465-482.

Lofti, Z. M., and A. J. Tosser. "Inhibited Expressions of Boolean Differences." International Journal of Electronics, 48-1 (1980), 57-64.

MacDonald, F "LSI Changes Basic Test Rules and Strategies." Electronics Test (1978), 59-60.

Mano, M. M. Computer System Architecture. Englewood Cliffs, N.J. Prentice-Hall, Inc., 1976.

Marinos, P. N. "Derivation of Minimal Complete Sets of Test-Input Sequences Using Boolean Differences." IEEE Trans. on Computers, C-20 (1971), 25-32.

Muehldorf, E. I. "Designing LSI Logic for Testability." Digest of Papers. Semiconductor Test Symposium, 1976, 45-49.

Muth, Peter. "A Nine-Valued Circuit Model for Test Generation." IEEE Trans. on Computers, C-25, 6 (1976), 630-636.

Nanda, N. L., and R. G. Bennetts. "Reconvergence Phenomenon in Synchronous Sequential Circuits." Electronics Letters, 16-8 (1980), 303-304.

Pomeroy, Bruce A. "Automatic Stimulus Generation for Digital Testing." Wescon Tech. Papers, W. Elect. Show & Conv., 19 (1975), 1-6.

Powell, Theodore J. A Module Diagnostic Procedure for Combinational Logic. Final Report AD688-743. Urbana: Illinois Univ. Urbana Coordinated Sci. Lab., 1969.

Prior, A. C., and R. G. Bennetts. "Application of the Boolean-Difference Technique to Sequential Logic." Electronics Letters, 10, 23 (1974), 486-488.

Rai, Suresh, and K. K. Aggarwal. "An Efficient Method for Reliability Evaluation of a General Network." IEEE Trans. Reliability, R-27, 3 (1978), 206-211.

Ramanoorthy, C. V., and W. Mayeda. "Computer Diagnosis Using the Blocking Gate Approach." IEEE Trans. on Computers, C-20 (1971), 1294-1300.

Reed, I. S. "Boolean Difference Calculus and Fault Finding." SIAM Journal of Applied Math., 24 (1973), 134-143.

Roth, J. Paul. "Diagnosis of Automata Failures: A Calculus and a Method." IBM Journal of Research & Development, 10 (1966), 278-281.

Savage, John E. The Complexity of Computing. New York: John Wiley & Sons, 1976.

Savir, J. "Syndrome-Testable Design of Combinational Circuits." IEEE Trans. on Computers, C-29, 6 (1980), 442-451.

Schneider, R. R. "On the Necessity to Examine D-Chains in Diagnostic Test Generation." IBM Journal of Research & Development, 11 (1967), 114.

Sellers, F. F., M. Y. Hsiao, and C. L. Bearnson. "Analyzing Errors With the Boolean Difference." IEEE Trans. on Computers, C-17 (1968), 676-683.

Si, S. C. "Dynamic Testing of Redundant Logic Networks." IEEE Trans. on Computers, C-27, 9 (1978), 828-832.

Snethen, Thomas J. "Simulator-Oriented Fault Test Generator." 14th Design Automation Conference Proc., New Orleans, LA. (1977), 88-93.

Suresh, R., and K. K. Aggarwal. "A Computer Method for Fault Detection in Combinational Circuits." Int. J. Electronics, 47-3 (1979), 247-251.

Thayse, A., and Marc Davio. "Boolean Differential Calculus and Its Application to Switching Theory." IEEE Trans. on Computers, C-22, 4 (1973), 409-420.

Thomas, J. J. "Automated Diagnostic Test Programs for Digital Networks." Computer Design (1971), 63-67,

Williams, T. W.. and K. P. Parker. "Testing Logic Networks and Designing for Testability." Computer (1979), 9-21.

Wolfgang, J. Paul. "A 2.5N-Lower Bound on the Combinational Complexity of Boolean Functions." Proc. 7th Annual ACM Symposium on Theory of Computing (1975), 27-36.

Yau, S. S., and Y. S. Tang. "An Efficient Algorithm for Generating Complete Test Sets for Combinational Logic Circuits." IEEE Trans. on Computers, C-20 (1971), 1245-1251.

2

VITA

Charlotte Couch Acken

Candidate for the Degree of

Doctor of Philosophy

Thesis: A MATRIX MODEL OF DIGITAL SYSTEMS AND ITS APPLICATION TO AUTOMATIC TEST GENERATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Harrison, Arkansas, August 10, 1947, the daughter of Mr. and Mrs. William H. Couch.

Education: Graduated from Bruno High School, Bruno, Arkansas, in May, 1964; received the Associate of Science degree from the College of the School of the Ozarks, Point Lookout, Missouri, in May, 1966; received the Bachelor of Science degree in Mathematics from the University of Arkansas, Fayetteville, Arkansas, in January, 1969; received the Master of Science degree in Mathematics from the University of Arkansas, Fayetteville, Arkansas, in January, 1971; received the Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma, in July, 1977; completed requirements for the Doctor of Philosophy degree at Oklahoma State University, in May, 1982.

Professional Experience: Instructor of Mathematics, Mathematics Department, University of Arkansas at Monticello, Monticello, Arkansas, 1970-1976; Member of Technical Staff, Sandia National Laboratories, Albuquerque, New Mexico, 1978 to present.

Professional Organizations: Member of Pi Mu Epsilon, Eta Kappa Nu, and Phi Kappa Phi Honor Society; former faculty member of the Mathematics Association of America.