TEST GENERATION AT THE TRANSISTOR LEVEL FOR MOS COMBINATIONAL LOGIC CIRCUITS

By

MOHAMMAD TAGHI MOSTAFAVI

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1980

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1982

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, 1986

Thesis
1986D
M915t
cop. 2

TEST GENERATION AT THE TRANSISTOR LEVEL

FOR MOS COMBINATIONAL LOGIC CIRCUITS

Thesis Approved:

[illegible]
Thesis Adviser

Bennett Basore

David L Soldan

[illegible]

[illegible]
Dean of the Graduate College

ACKNOWLEDGEMENTS

My sincere appreciation is extended to Dr. Louis G. Johnson, my thesis adviser, for his invaluable assistance throughout my graduate education in electrical and computer engineering. I am grateful to Dr. Bennett L. Basore, Chairman of my doctoral committee, and other members of my doctoral committee, Dr. David L. Soldan and Dr. Michael J. Folk, for their support and for their encouragement and friendship. The typing of the final copy was performed by Betty L. Harris, to whom I am most grateful.

Well deserved gratitude is extended to my parents and other members of my family for their loving care and their financial support. I thank my four year old son, Behrooz, for his patience during the time that I have been away from home working on this thesis and for not being able to spend more time with him. I am thankful for my lovely wife, Zary, 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

Many algorithms have been proposed and some have been programmed for generating tests for logical faults of digital circuits at the logical gate level. At the logical gate level the circuit is considered to consist of elements whose input(s) and output assume only the values denoted by logic 0 and 1. The basic elements of the circuits at the logical gate level are called gates. At the transistor level, on the other hand, elements in the circuits are switches (positive and negative) whose input and output are not restricted to logic 0 and 1, but also could be logic Z (high impedance).

Usually more than one switch, at the transistor level, represents a gate at the logical gate level; therefore, the operation of the gates and the switches are not the same for a particular circuit. Any logic function can be represented by gates. However, a gate level diagram may not accurately represent the way the circuit is implemented at the transistor level. The pass transistor (transmission gate), which is an important element in MOS digital circuits, does not have an equivalent gate at the logical gate level. Contacts at the logical gate level and the transistor level of a particular circuit can be defined differently. Thus, the logical gate level representation does not accurately describe or identify the necessary topological information of MOS type circuits for test generation purposes.

The common fault model used in digital circuit testing at the logical gate level is the single stuck-at model. The possible faults in this model are either stuck-at logic value "1" (s-a-1) or stuck-at logic value "0" (s-a-0). In some MOS circuits, stuck-open faults also need to be detected, which is not easy or is impossible using one of the test generation techniques at the logical gate level. For example, the stuck-open faults in 2-input CMOS "NAND" and "NOR" gates described in Chapter III might not be detected by using the gate level generated tests. However, the stuck-open faults can be detected at the transistor level representation of MOS circuits. The stuck-at model at the logical gate level is not designed to handle short-circuit faults. In some particular circuits, it might be possible to find the corresponding line (node) in the logical gate level representation of the actual circuit and use the stuck-at model to detect the short circuit faults. However, in some cases, it is quite difficult or impossible to find the corresponding line (node) at the logical gate level of a particular circuit. The short-circuit faults can be handled more easily by using the transistor level implementation of the MOS circuits. In some cases, by using the structural properties of the MOS circuits (i.e. using transmission gates to implement multiple paths in a circuit), it is possible to design a much smaller test set at the transistor level rather than the gate level. Overall, test generation methods available for the logical gate level are not completely adequate to generate tests for every MOS circuit. Therefore, a new model or technique is needed to generate tests at the transistor level implementation of MOS type design.

In the past few years many problems involving testing MOS circuits have been considered and some solutions have been proposed [1], [2], [23], [6], [24], [25]. The proposed solutions, for testing purposes, view the MOS circuits at the conventional gate level for some specific circuits. Some of the solutions are designed to remodel the structural properties of the system at the circuit level; these take away some of the advantages of MOS-circuit design methods including simplicity and small size. Other solutions, if possible, deal with remodeling the corresponding logical gate level of the actual circuit which is quite difficult. The proper way to generate tests for MOS digital systems is at the transistor level of the circuits.

The existing CSA (connectors-switches-attenuator) switching theory implements the structural properties of any MOS digital circuit at the transistor level [14]. Switches in the CSA networks are bidirectional, which makes defining a path from the input(s) of the circuits to their output(s) quite difficult. Because of bidirectional switches, a path which is started from one input in CSA switching theory may end at the same input (looping) or other inputs rather than an output.

This research presents a modified version of the CSA switching theory called wGS (wired-Gates-Switches) network. The wGS network implements the structural properties of any MOS digital circuit for testing purposes. In the wGS networks, every line is single directional and there is at least one path from every input to an output of a circuit. The path sensitization method, which is a known algorithm for generating tests at the logical gate level of digital systems, can be used on the wGS network to generate tests at the transistor level of MOS circuits. Based on the existing test generation method at the logical gate level, a new algorithm called "Critical Path Test Generation at the Transistor Level" (CPTGTL) is proposed and programmed. CPTGTL uses the wGS network to generate tests at the transistor level of MOS combinational circuits.

Chapter II is a review of some of the best known and most widely used gate level test generation methods. These methods are compared, and their advantages and disadvantages are pointed out. The CSA switching theory is also described in Chapter II. It is shown that the CSA switching theory implements the structural properties of MOS digital circuits.

Some examples are used in Chapter III to examine gate level generated tests on transistor level MOS circuits. The advantages and necessity of testing the MOS-type design at the transistor level rather than the logical gate level are explained.

The "wired-Gate-Switches" network, the modified version of CSA switching theory, which defined direction in every line throughout the transistor level implementation of the MOS circuits is introduced in Chapter IV. The wired-gate elements which play the central role in the logic behavior of the the circuit models, other elements, and definitions and rules to describe the wGS networks are also pointed out.

Critical Path Test Generation method at the Transistor Level along with some definitions and rules to describe the algorithm is introduced in Chapter V. Beside all of the related single stuck-at faults at the logical gate level, the test set generated by CPTGTL detects single stuck-open faults in transistor-level implementations of MOS digital circuits. Several examples are included to illustrate concepts and techniques.

Chapter VI contains the conclusions and suggestions for further research in the area of test generation at the transistor level of MOS type digital circuits. These include developing a method to preprocess a transistor level MOS circuit description to automatically provide wGS inputs to the existing program along with a method for generating tests for short circuit faults. But more important, a design method for testing redundant MOS circuit at the transistor level is introduced.

Besides the six chapters, two appendices are also included in this research. Appendix A contains the listing of PL/1 programs which implement the CPTGTL algorithm. Appendix B contains a few wGS network implementations of some MOS circuits, the listing of the network inputs, and the outputs (generated tests) for these networks.

CHAPTER II

REVIEW OF RELATED LITERATURE

Test Generation

After a circuit has been designed and manufactured, one must apply some kind of stimuli to the circuit and check its responses to determine if it is functioning properly. Depending on the circuit to be tested, the economic situation, and the equipment available, different approaches can be taken for stimulus generation and response checking.

One method of testing is to apply every possible input combination to the combinational circuit, usually with a counter, and check the output(s). This is called the exhaustive method or exhaustive test generation. Depending on the size of the circuits a large number of test patterns may be required; the exhaustive method is feasible only for small circuits. Another method uses a long sequence of pseudo-random input(s) and checks for proper output(s); this method will not test every possibility. The third and best of these methods for testing is the generated test patterns which have been precomputed and stored before the test run [3], [4], [5]. The generated test patterns can be developed either through simulation or by using one of the test generation techniques which are of main concern in this report.

The available methods for test generation that employ an explicit circuit model reflecting both the structure and behavior of the digital systems will be examined. These models are generally designed at the logic level (gate level) with elements (gates) such as AND, OR, NAND, NOR, NOT, and EXCLUSIVE-OR. But recent concern is centered around modeling those aspects of MOS IC's such as transmission gates (pass transistors) which do not correspond to standard logic gates [6], [7].

One of the common fault models used in digital circuit testing is the single stuck-at model. In this model each circuit node is assumed to have two potential faults which are permanently stuck-at logic value "1" (s-a-1) or stuck-at logic value "0" (s-a-0). Although the single stuck-at model does not adequately portray every possible fault in a circuit, experience has shown that it will cover more than 80 percent of the faults in a particular circuit; therefore, the single stuck-at model produces a result within the acceptable defect levels for that circuit. Other fault models such as multiple stuck-at faults, delay faults and short-circuit faults have received less attention. However, it can be shown that testing a circuit at the transistor or layout level more easily handles short-circuit faults.

Test Generation for Combinational Circuits

Algebraic Methods:

Algebraic methods use circuit equations to generate tests and do not consider electrical connectivity. An example of such approaches is the Boolean difference method [8], [9].

A combinational circuit can be expressed as a function $F(x_1, x_2, \ldots\ldots\ldots, x_n)$. If a fault causes one of the inputs to s-a-0, the faulty circuit then can be shown as the function

$F^*(x_1, \ldots\ldots x_n) =$

$$F^{*}(x_1,\ldots x_{i-1},0,x_{i+1},\ldots x_n)$$

F* shows the function of faulty circuit.

To detect such a fault, one needs to select input values to test line "i" which can be derived from its Boolean differences defined as

$$dF/dx_i \overset{\Delta}{=} F(x_1,..x_{i-1},1,x_{i+1},\ldots x_n) \oplus F(x_1,..x_{i-1},0,x_{i+1},\ldots x_n)$$

The above function, which is the Boolean difference of function F*, represents all conditions such that a change in the value of x_i will cause a change in the output value of the circuit. As an example, in Figure 1 the output of the circuit is defined by the Boolean expression $f = ((x_1+x_2)x_3 + x'_3x_4)$.[1] To detect the fault x_3 "s-a-0" or "s-a-1", first we find dF/dx_3 that is

$$dF/dx_3 \overset{\Delta}{=} F(x_1,x_2,0,x_4) \oplus F(x_1,x_2,1,x_4) = x_4 \oplus (x_1 + x_2) = x'_1x'_2x_4 + x_1x'_4 + x_2x'_4.$$

Then the set of tests that detects the fault x_3 s-a-1 is defined by the Boolean expression

$$R = x'_3(dF/dx_3) = x'_3(x'_1x'_2x_4 + x_1x'_4 + x_2x'_4).$$

The set of tests for x_3 s-a-0 is defined by the Boolean expression $R = x_3(dF/dx_3)$. The set of tests which detect x_4 s-a-0 is defined by the Boolean expression

$$x_4(dF/dx_4) = x_4(x_3(x_1 + x_2) \oplus (x_3(x_1 + x_2) + x'_3)) = x'_3x_4.$$

And for x_4, s-a-1 will be the Boolean expression $x'_4(dF/dx_4) = x'_3x'_4$.

---

[1] Symbol "'" represents "The opposite of." Read "x'" as "x bar."

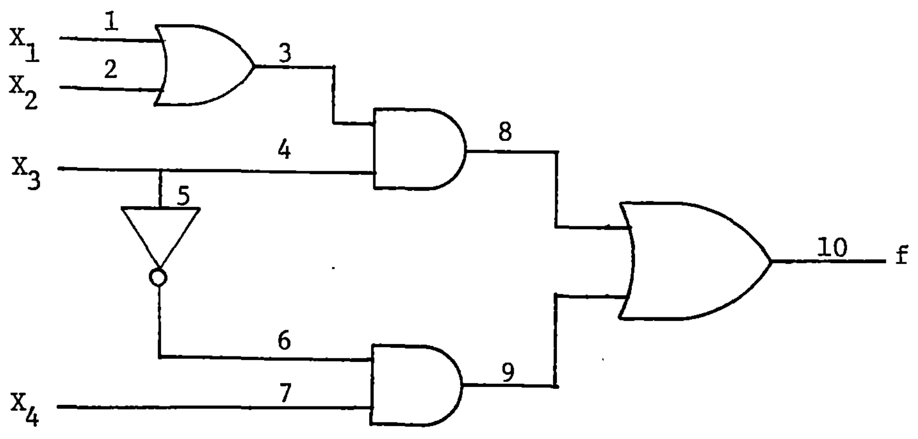


Figure 1. A Gate Level Structure of Digital Circuits

Path Sensitization and the D-Algorithm

The test generation procedures considered in the previous discussion may be characterized as algebraic or functional in that they utilize circuit equations to generate tests. Now we consider the path sensitization technique which consists of two basic activation and propagation steps.

To detect the fault "a", s-a-0 (s-a-1), a test must cause the signal "a" to take the opposite value, i.e., 1(0), in the normal (fault free) circuit; this is the activation step which is a necessary but it is not a sufficient condition to detect the fault. The effect of the fault, the error signal from the site of the fault, must then be propagated along some path all the way to an output and activated by backtracking along another path all the way to an input. To complete the test, one needs to do implication on other gate inputs (line justification) which involves specifying additional input values needed to sensitize the path using a backtracking approach from each sensitized node to input(s) of the circuit [10].

The path sensitization approach can be formalized in a form that enables us to use it in computers; this has been done and named the D-Algorithm [11], [12]. The D-Algorithm is a method for generating a test vector for a given fault. It is typically used with gate-level circuit models and stuck faults. Symbol D represents a signal which has the value 1 in the normal circuit and 0 in the faulty circuit (D' is 0 in the normal circuit and 1 in the faulty circuit). Each element (gate) has its own function redefined in terms of the symbols D and D' as shown in Figure 2. To show how the D-Algorithm works, let us go back to Figure 1. To test line 2 s-a-0, we start with D on line 2. The three propagation steps from line 2 to line 3 to line 8 to line 10 can be tabulated as shown in Figure 3.

The D is propagated through each elementary gate, one by one in turn, from one input (line 2) to the output (line 10) without regard to the state of other gates. The implication steps assign logic values (0,1,1,0,X,0) to other circuit lines; logic value X represents "don't care" which can be either 0 or 1. For example, in order for line 8 to take on the Value D, line 4 must be logic 1. Line 4 being 1 implies line 5 is also logic 1. To propagate D from line 8 to line 10, line 9 must be logic 0. Line 9 being 0 implies either line 6 or line 7 or both being 0. But line 6 being logic 1 implies line 5 to be logic 0 which contradicts the previous assumption. Contradictions such as a line taking on both a 0 and a 1 value indicate the nonexistence of that test. Therefore, line 6 must be logic 0 and 7 "don't care" (logic X).

AND			OR			NOT	
In-1	In-2	Out	In-1	In-2	Out	Input	Output
1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1
0	1	0	0	1	1	D	D'
0	0	0	0	0	0	D'	D
1	D	D	1	D	1		
D	1	D	D	1	1		
1	D'	D'	1	D'	1		
D'	1	D'	D'	1	1		
0	D	0	0	D	D		
D	0	0	D	0	D		
0	D'	0	0	D'	D'		
D'	0	0	D'	0	D'		
D	D	D	D	D	D		
D'	D'	D'	D'	D'	D'		
D'	D	0	D'	D	1		
D	D'	0	D	D'	1		

Figure 2. Elementary Gate Function Defined By Symbol D

Critical Path Test Generation

The test generation techniques that have been considered above are fault-oriented in that they attempt to derive tests for each specific fault. We now consider a method which derives tests to detect many faults at the same time, faults and tests are independent of each other. One test generation procedure is a random test generation which is fault independent and has been explained before. The other is a procedure called Critical Path Test Generation; it is intended to detect many struck-type faults simultaneously. Critical path test generation is based on the observation that half of the stuck type faults along a sensitized path are detected by a single test. For example, in the circuit shown in Figure 4 to test for "e" s-a-0, the test method shown above detects the fault "a" s-a-1, "e" s-a-0, and "g" s-a-0, all of which lie on the indicated sensitized path.

Many faults can be detected simultaneously by maximizing the number and length of the sensitized paths produced by a given input pattern.

> Such a test is produced using the concept of sensitized cubes. Starting from the output, whose value is defined as 0 or 1, a path is traced backwards toward the input. At each logic element, if the output value has been specified, the element input values are specified in such a way that the value of one or more of these inputs is critical (i.e., if the value of the critical input changes, the value of the element output will change) [10].

For example, for 2 input "NAND" gate with inputs "a" and "b", and output "c" for output c=1, the input a=0, b=1 has "a" as a critical input and a=1, b=0 has "b" as a critical input. Lines a=b=0 have no critical inputs. But for output c=0 both a and b have critical inputs with value 1(a=b=1).

Step	Line 1	2	3	4	5	6	7	8	9	10
Initial test on Line 2	X	D	X	X	X	X	X	X	X	X
Propagate to Line 3	X	D	D	X	X	X	X	X	X	X
Implication on other gate inputs	0	D	D	X	X	X	X	X	X	X
Propagate to Line 8	0	D	D	X	X	X	X	D	X	X
Implication on other gate inputs	0	D	D	1	1	X	X	D	X	X
Propagate to Line 10	0	D	D	1	1	X	X	D	X	D
Implication on other gate inputs	0	D	D	1	1	0	X	D	0	D

Figure 3. Forward Propagation and Implication

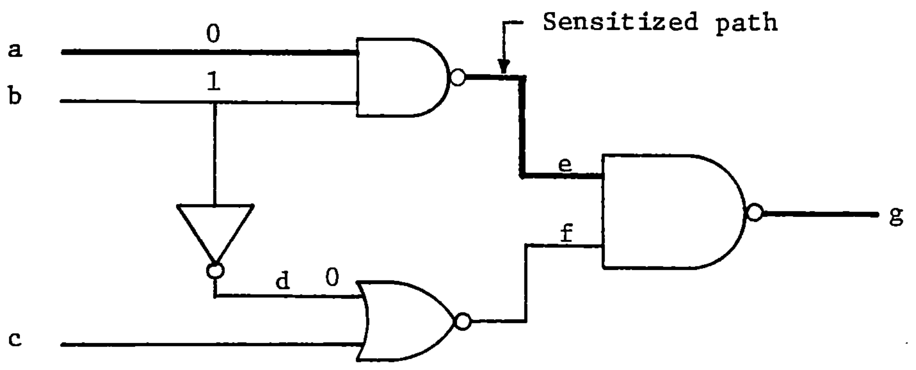
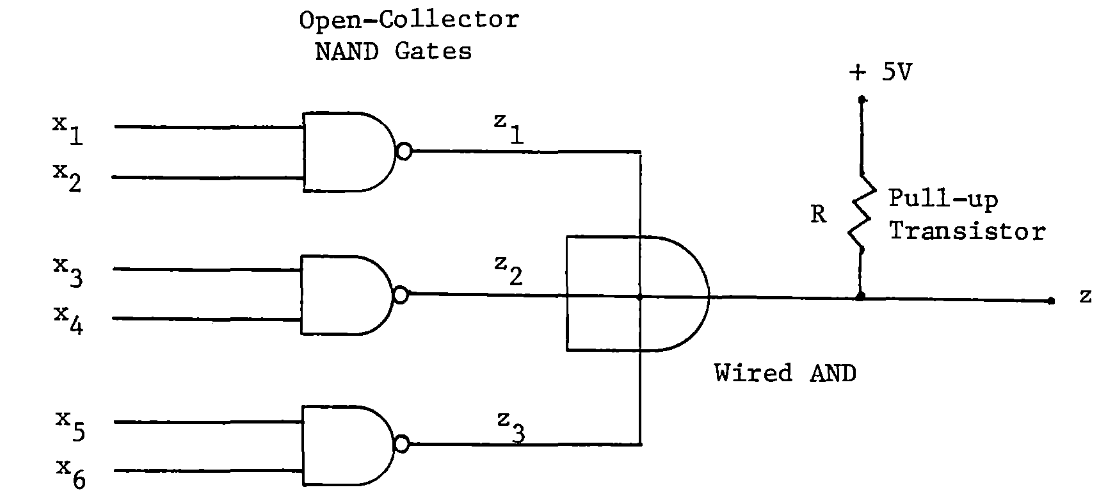


Figure 4. A Gate Level Structure Using Sensitized Path

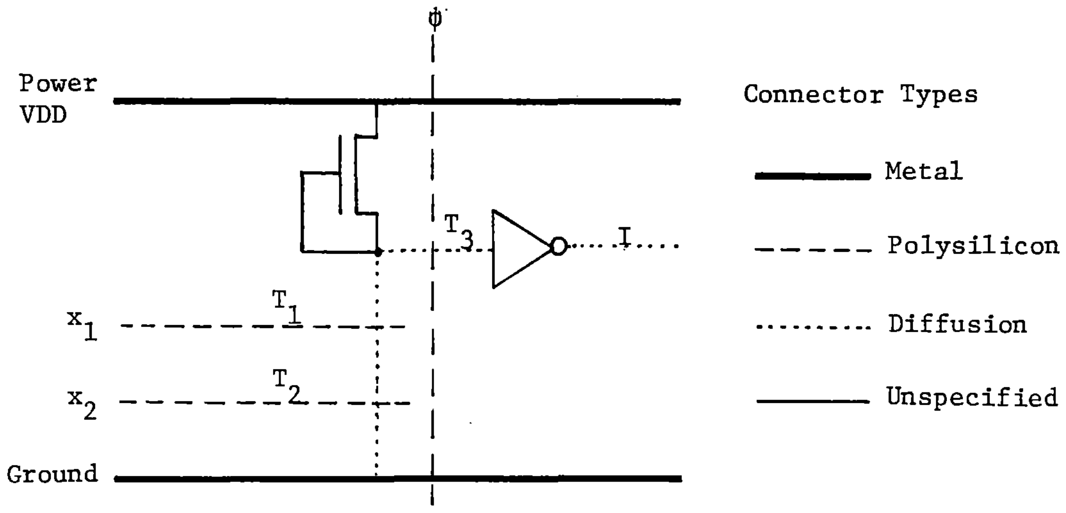
Switching Theory for MOS Circuits

Switching Theory was defined by Shannon [13] in the 1930's to provide a mathematical framework for analysis and synthesis of digital systems at the logical or binary level [14]. This theory has been under development for the past forty to fifty years [15], [16]. In the 1940's and 1950's work was concerned with contact network or electromechanical relay circuits. A contact is a bidirectional switching with "ON" and "OFF" modes. In the 1960's, attention moved to gate networks which were intended to model electronic switching such as vacuum tubes or transistors (Boolean switching function). The assumption was that contact and gate networks were the only possible types of switching [15], [17]. But in modern digital systems, particularly MOS circuits, switching theory is essentially different from either a classical contact or gate switch.

The logic operation of most circuits is dependent on the elements that, under certain conditions, will pull a signal up (logic 1) or down (logic 0). In some MOS circuits contact and gate networks have been combined in a way that cannot be modeled by either contact or gate network alone. Connectors appearing in a logic diagram do not necessarily correspond to those in physical layout diagrams. Classical theory only recognizes the two logic values 0 and 1, but modern logic circuits use other values such as high-impedance (Z). Power and ground are omitted in the logic gate level, but they are part of the layout level. There are a lot of other reasons available that show the difference between switching theory of logic gate level and layout level of MOS circuits [15], [17], [18]. As a result, switching theories



(a) A Bipolar Circuit containing Wired Logic [18]



(b) An NMOS Circuit containing IC Layout Information [18]

Figure 5. Gate and Mixed Circuits

of contact and gate networks will cover part of the modern digital system (MOS circuits). Because the logic diagrams are a limited value of MOS type design, other switching theories for MOS circuit in the layout level are needed.

Fortunately, in recent years several programs have been introduced to solve the above problems. An example is LOGIS (one MOS-oriented simulation program) which allows MOS pull-up devices and wired logic gates to be treated as primitive components [19]. Also, the work of various researchers [20], [21], has led to a reappraisal of switching theory in the MOS context [14].

One result is a new theory based on the switching of the MOS circuit element called CSA (Connector-Switch-Attenuator) theory [20], [22]. CSA theory allows for most kinds of switching circuits including contact, gate and mixed circuits as shown in Figure 5.

The fundamental components of the circuits in the new model are connectors, switches (representing transistors), and attenuators (representing pull-up/down load elements). The logic behaviors defined by MOS switching theory are digital values 0 (values denoting signals in the low-voltage), 1 (values denoting signal in the high-voltage), Z (values denoting high-impedance or disconnected state), and U (values denoting unknown or indeterminate state). Logic values can be increased for more accuracy near the desired level which will be explained later. Thus, CSA switching theory can be used as a good description for MOS circuits [14].

CS Networks

CS networks are combined with switches and connectors. Switches are the basic components of any switching circuit, and connectors are lines that interconnect the switches. Figure 6 shows the connector and switches used in CS switching theory and also their represented IC elements.

A switch S is a three-terminal device in which K is the control terminal and D_1 and D_2 are symmetric data terminals. When S is "ON" (K has logic "1" for positive switches and logic "0" for negative switches), D_1 and D_2 are connected together. When S is "OFF" (K is logic "0" for positive and logic "1" for negative switches)[1], D_1 and D_2 are disconnected. Connectors in CSA network play a central role in logic behavior of the circuit models; joining of two or more connectors performs a wired-logic operation. Figure 7 shows the logic behavior of connector C in terms of logic values V4=(0,1,U,Z).

The logic operation of a connector is denoted by # (i.e., for Figure 7, output $c = \#(x_1,x_1)$. In CS network, as we see in Figure 7, logic 0 is stronger than logic Z which is denoted as 0>Z; similarly, 1>Z, U>1, and U>0. There is no definite relation between logic 1 and 0. In other words, we cannot say that 1>0 or 0>1; the result of connecting a logic signal 1 to a logic signal 0 is dependent on how strong those signals are. To know exactly what the result of connecting logic 1 to a logic 0 is, we need to define a different level of 0 and 1 logic signals.

---

[1] In this report K is assumed to have only a logic value of either 0 or 1.

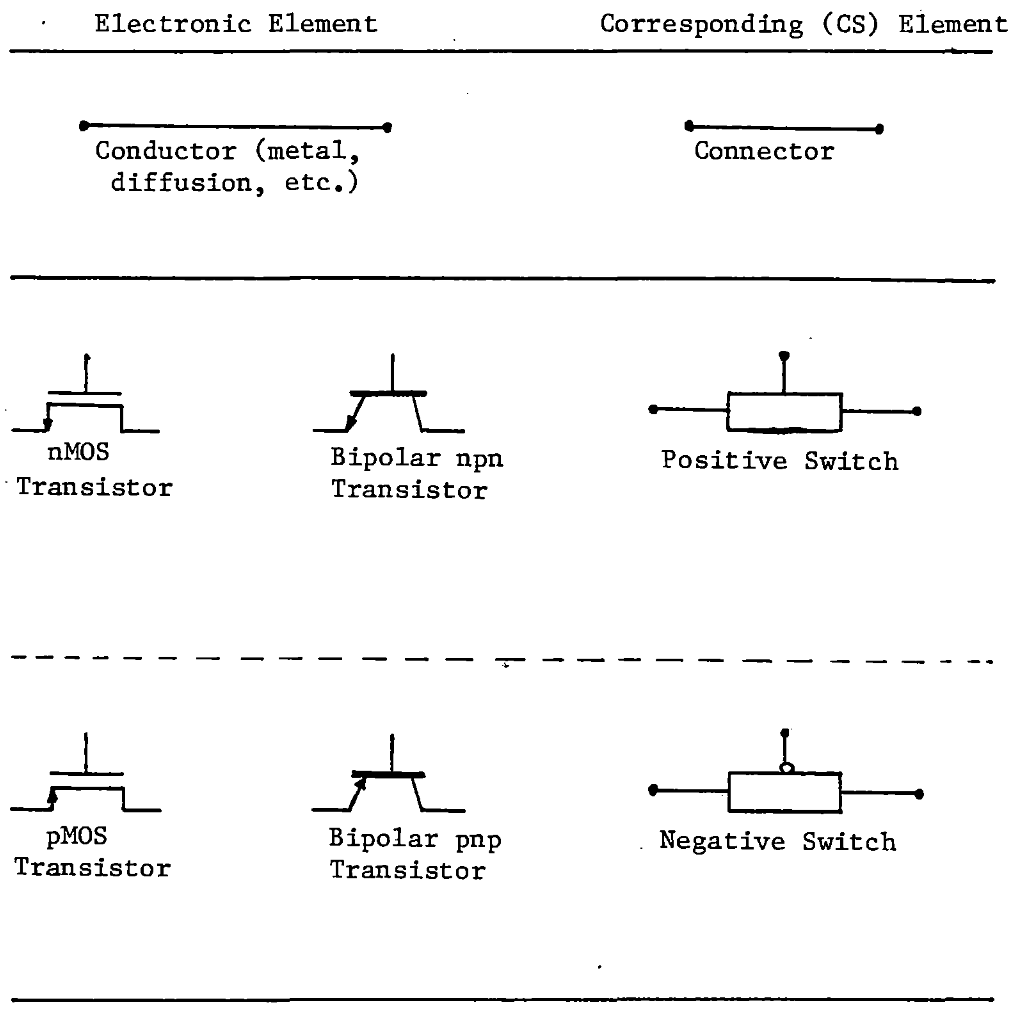
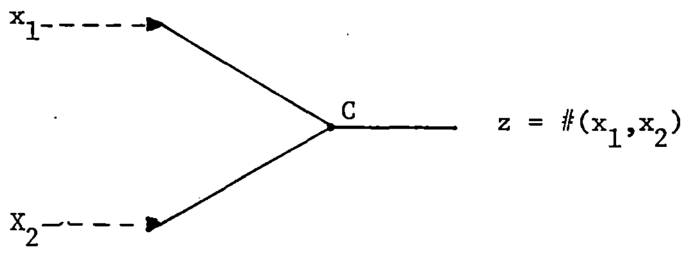


Figure 6. The Connector and Switch Model used in CS Network [14]



x_1	x_2	z
0	0	0
1	1	1
Z	Z	Z
0	Z	0
Z	0	0
1	Z	1
Z	1	1
0	1	U
1	0	U
U	d	U
d	U	U

Figure 7. The Behavior of a Connector C with respect to V4=(0,1,U,Z), d denotes "don't care" value [14]

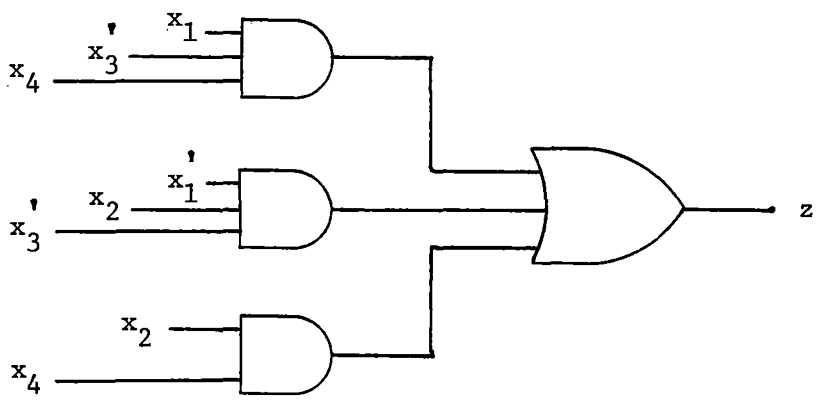
Definition 1: let C be a connector to which the input signals x_1, $x_2, \ldots, x_k$ V4 are applied. C realizes a <u>connection function</u> # that produces a unique output signal $z_c = \#(x_1, x_2, \ldots, x_k)$, where z_c is the logically weakest member of V4 such that $z_c >= x_i$ for $i=1,2,\ldots,k$ [14, p. 1142].

Relation >= means "is of equal or greater logical strength than." Figure 8 shows a gate network, a contact network, and a CS network that models the contact network of Figure 8(b). A contact network (i.e., Figure 8(b)) can be modeled by a CS network (i.e., Figure 8(c)) precisely, if there is one-to-one correspondence between the connectors and contact of Figure 8(b) and the connectors and switches of Figure 8(c).

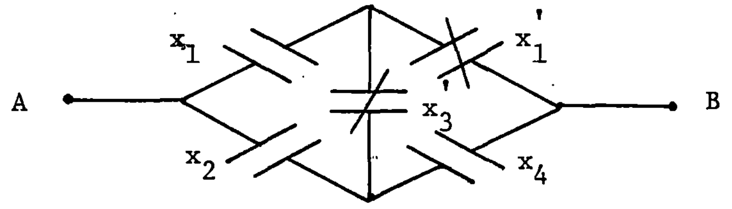
Definition 2: A CS network N is called a (two-terminal) CS contact network if it satisfies the following constraints:

1) N has exactly two external input-output terminals A and B.
2) An input signal x_i or X_I is applied to a terminal T of N if and only if T is a control input to a switch.
3) Only terminals from A, B, and the set of switch data terminals are joined by connectors [14, p. 1142].

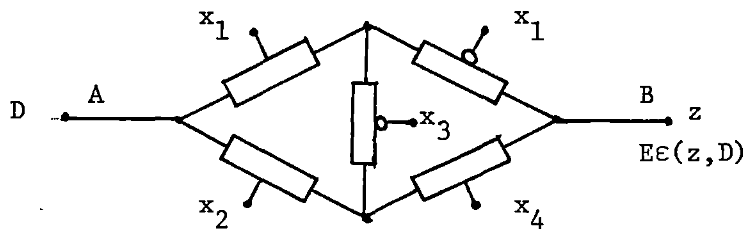
Considering the second definition and Figure 8(c), the Boolean function "z" realized by a contact network assumes the value 1 whenever A and B are connected and the value 0 when they are not; "z" is usually called a <u>transmission function</u> [15]. The Boolean function "z" in a CS network, on the other hand, is one of four logic V4=(0,1,U,Z). If logic value 0(1) is applied to one external terminal A of CS network N_1 as shown in Figure 8(c), the output function in the terminal B is (0(1),Z). Therefore, to reach the proper logic value (0 or 1) we need



(a) A Gate Network



(b) A Contact Equivalent Network



(c) A CS Equivalent Network

Figure 8. A Gate, Contact, and CS Network

to map terminal B of the network from (0(1),Z) to 0(1); this mapping is also necessary if CS networks are to be interconnected.

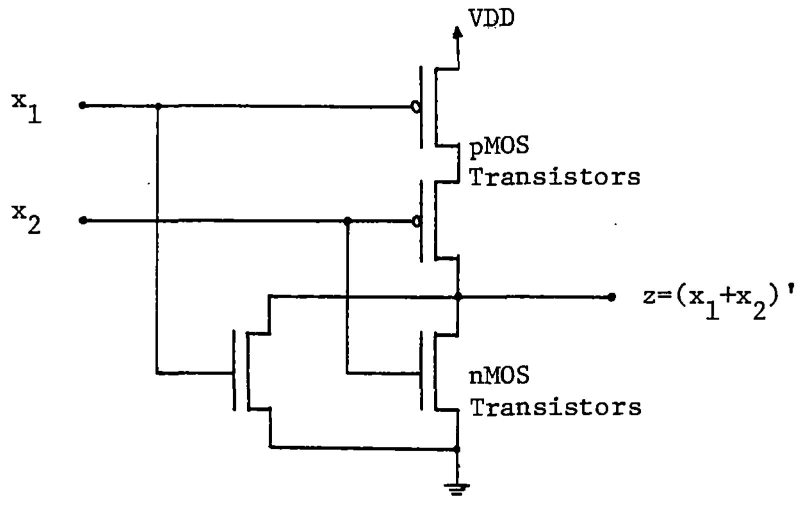
One solution is to design another CS network "N_2" to give desired output logic 0 or 1 when the output terminal of N_1 is logic Z, and to make contact with the output terminal of N_1 and N_2. Then the output terminal of N_2 will force the output terminal of N_1 to be 0 or 1. For example, Figure 9(a) shows a typical CMOS circuit for a two-input NOR gate, and Figure 9(b) shows the previously discussed method of CS network equivalent to Figure 9(a); this special type of CS network is called a complementary CS network. Figure 10 shows another way of using CS network in design [22]. Figure 10(a) is an NMOS selector network, and 10(b) is the equivalent CS network.

CSA Network

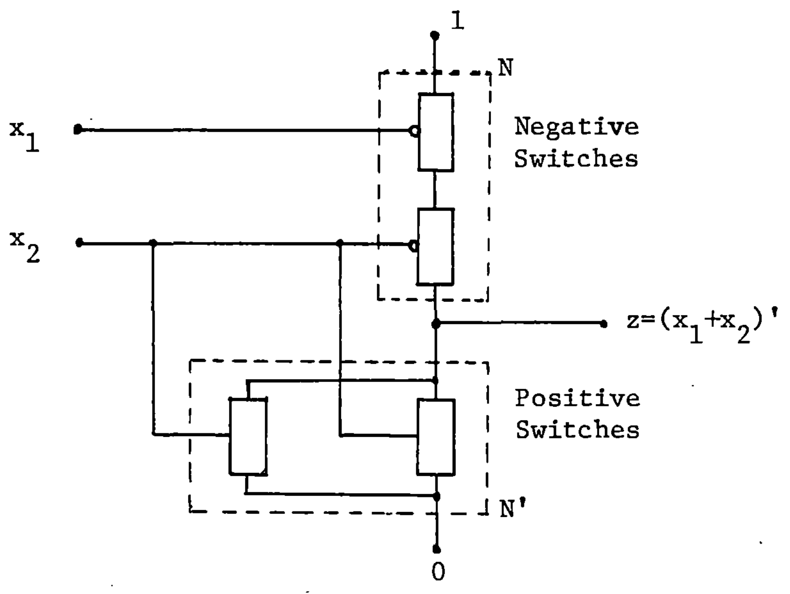
It was realized that it is necessary to be able to convert undesired Z signals to 0 or 1. The complementary and selector CS network of Figures 9 and 10 provide two solutions and can perform this conversion; however, each has certain limitations. Complementary networks are suitable for only CMOS type circuits (circuits with both pMOS and nMOS type transistors), and selector networks cannot perform logical inversion.

A solution to the above difficulties is another device usually called a load, that can "pull-up" Z to logical 1 or "pull-down" Z to logical 0. In MOS circuits, pull-up load can be a resistor or a load transistor [14].

A CSA network is a combination of CS network(s) and one or more new devices called attenuators which are equivalent to pull up/down loads

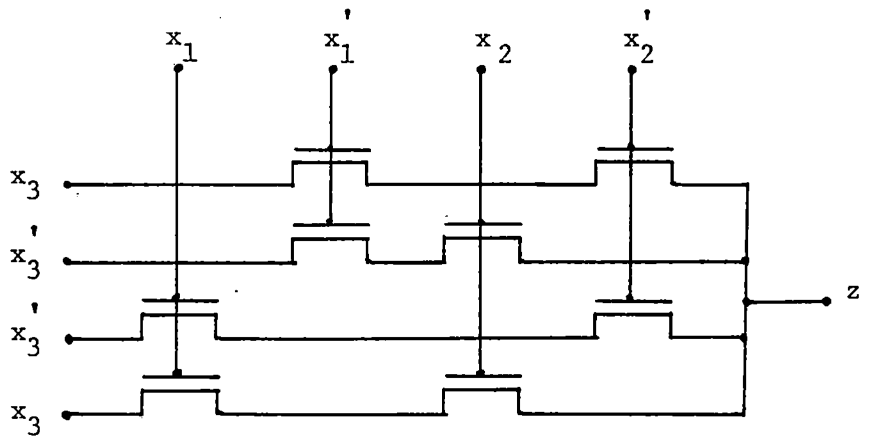


(a)

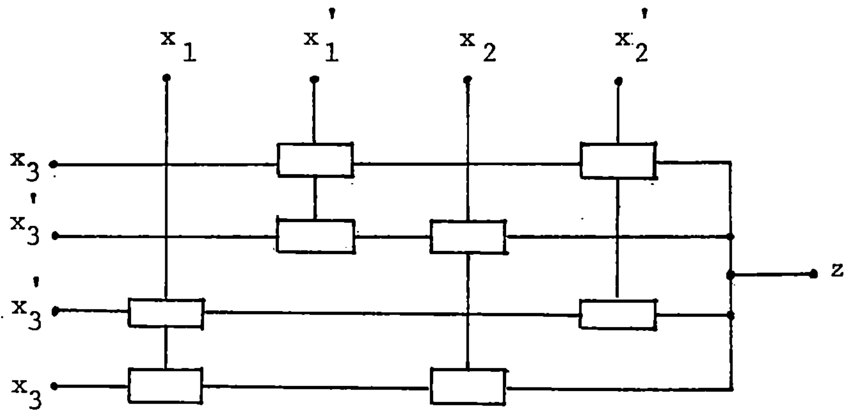


(b)

Figure 9. A CMOS "NOR" Gate (a) and the Equivalent Complementary CS Network (b) [14]



(a)



(b)

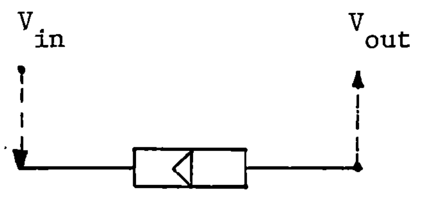
Figure 10. An NMOS Selector Logic Circuit (a) and the Equivalent CS Network (b) [14]

in MOS circuits. The problem of having output logic Z in a CS network can be solved by adding one or more attenuator devices to the network. An attenuator is a two terminal device rather than a three terminal. The Boolean function of an attenuator, assuming output of the attenuator is connected to an output terminal with value Z, can be defined as follows:

1) Pull-down attenuator
   a) z = #(Z,Q) = 0
   b) z = #(1,Q) = 1.
2) Pull-up attenuator
   a) z = #(Z,Q) = 1
   b) z = #(0,Q) = 0.

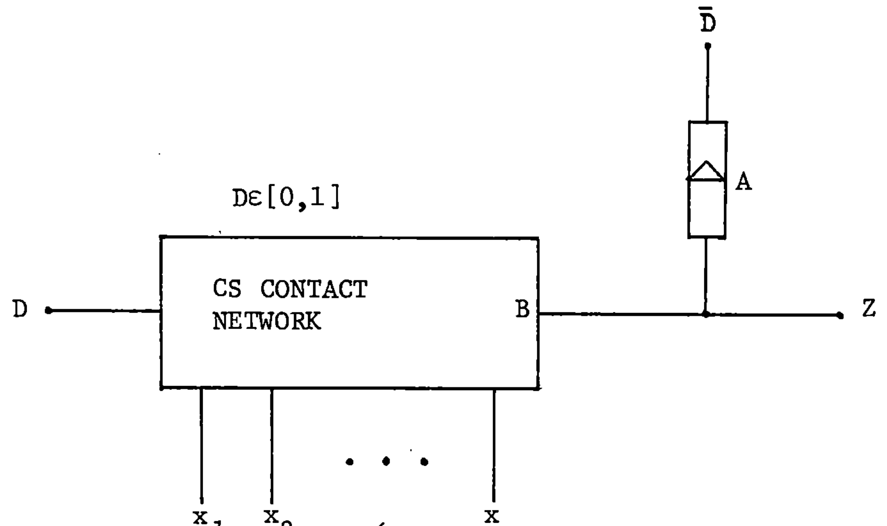
Q is the defined output of the attenuator whenever there is a Z signal that needs to be mapped to 0 or 1, and z is the resultant output.

The value of Q cannot be either logic value 0 or logic value 1 because, for example, with the value Q=1 under pull-down attenuator, this result will be proper in part b, but not in part a; under pull-up attenuator, the value Q=1 will be proper in part a, but not in part b. The solution is to define new logic values besides V4=(0,1,U,Z). Proper new logic values for attenuator are "weak" values of 0, 1 or, to be more precise, "weak" values of 0, 1, and U [18], [19]. The "weak" values of 0, 1, and U can be overridden by "strong" values of 0, 1, and U respectively, and all of them override Z. Therefore, in CSA networks instead of having V4=(0,1,U,Z), the possible logic values could be V7=(0,1,U,Z,"weak"0,"weak"1,"weak"U). Figure 11 shows an attenuator, a truth table for an attenuator device, and a CSA gate model combined with a <u>CS Contact Network</u> and an <u>Attenuator</u>.



V_{in}	V_{out}
0^w,0	0^w
1^w,1	1^w
U^w,U	U^w
Z	Z

(a)



(b)

Figure 11. An Attenuator defined on V7(a), its Application to construct a CSA Gate (b). Symbol "w" represents the "weak of" [14]

CHAPTER III

EXAMINING GATE LEVEL GENERATED TESTS ON TRANSISTOR LEVEL MOS CIRCUITS

Introduction

There are many reasons to believe that tests generated at the logic gate level of digital systems are not always suitable for testing MOS circuits. In the past few years researchers have been giving considerable attention to finding the proper way of detecting faults in digital circuits [23], [6], [24], [25]. Thus many problems have been considered and some solutions have been proposed. Some of the solutions to the problem of testing MOS digital systems are: C/D tester for CMOS stuck-open-fault [23], modeling faults which are a peculiarity of CMOS digital integrated circuits structured at gate-level [6], modeling physical failures in MOS complex cells [24], modeling of special circuit structures at the gate and primitive level [25].

The above papers, for testing purposes, view the MOS circuits at the conventional gate level. Some of the proposed solutions deal with remodeling the structure of circuits that can then be tested at the corresponding gate level; other solutions remodel the corresponding gate level of MOS circuits. In some cases, the first method will take away some of the advantages of structural properties of MOS design including simplicity and/or size of circuits. The second method will complicate the corresponding gate level and make generating tests at the conventional gate level less efficient. A better solution is to generate tests using the transistor level of MOS circuits.

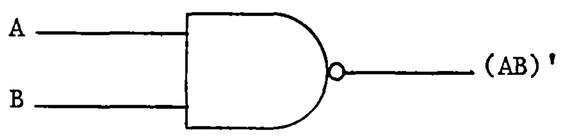
Since more than one transistor in digital circuits represents a logic gate, all of the functions represented by gates can simply be described by transistors, but the converse may not be true. The transmission gate (pass transistor) is an important element in MOS digital circuits and is used to represent a variety of logic functions. Pass transistor logic does not have an equivalent logic gate at the conventional gate level. In some cases, by taking advantage of the structural properties of the implementation of the MOS digital integrated circuits, it might be possible to come up with tests which are significantly smaller in number than the "stuck-fault" test set generated at the equivalent conventional gate level.

In this chapter it will be shown that there are some faults (failures) that can be easily detected using proper tests generated at the transistor level which are not easy or are impossible to find at the equivalent logic gate level of MOS circuits. However, the main purpose of this chapter is to show the very important advantage of testing MOS circuits at the transistor level; that is, in certain cases getting a test set which is much smaller than the "stuck-at" test set of the gate-level implementation.

Testing Circuits

Simple (Gate) Circuits

Figure 12 shows a 2-input NMOS "NAND" gate circuit diagram at the transistor level, a logic symbol and logic function (truth table).



A	B	(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

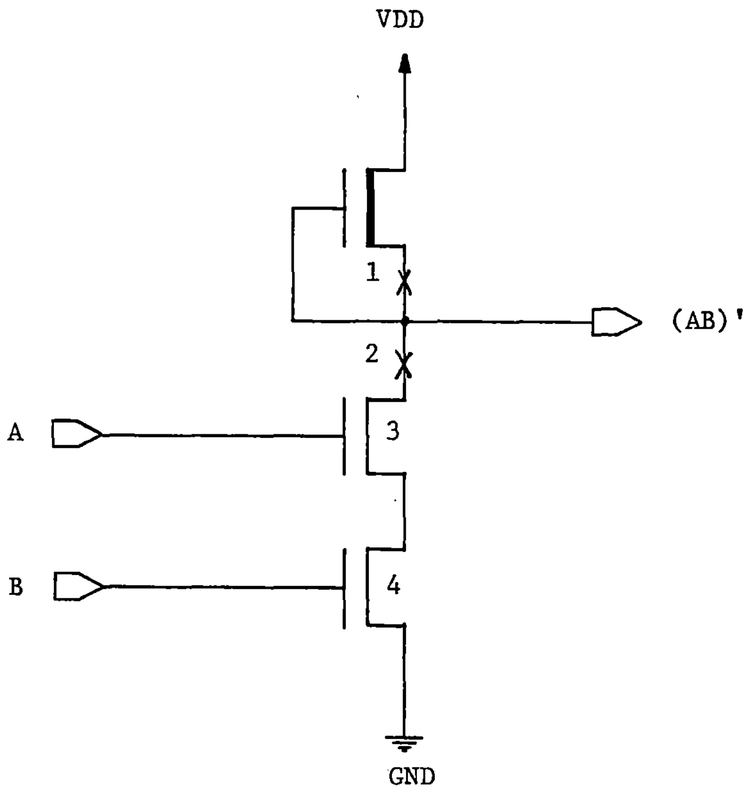


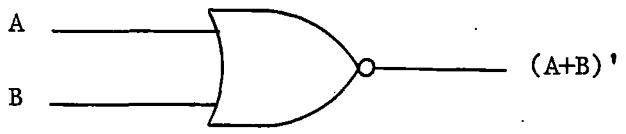
Figure 12. A 2-Input "NAND" Gate Logic Symbol, Logic Functions and Circuit Diagram

Various possible faults are marked with numbers 1 through 4. It should be noted that fault 2 indicates an "open" in the contacting path from ground to the output and fault 1 indicates an "open" in the contacting path from VDD to the output. Faults 3 and 4 are considered to be failures in transistors and contacts in interconnection lines. The "stuck-at" test set for the 2-input "NAND" gate with inputs A and B are 11, 10, and 01. It has been shown [26] that the "stuck-at" test set detects all faults in the circuit.

Figure 13 shows a 2-input NMOS "NOR" gate circuit diagram at the transistor level, a logic symbol, and logic function (truth table). Various possible faults are marked with numbers 1 through 6. The "stuck-at" test set for the 2-input "NOR" gate with inputs A and B are 00, 10, and 01 which detect all of the faults in the circuit.

Testing the CMOS equivalent of previous examples for 2-input "NAND" and "NOR" gate is different. Figure 14 shows a 2-input CMOS "NAND" gate circuit diagram at the transistor level.

The circuit in Figure 14 has been studied by Bashier [27] and it has been shown that besides the "stuck-at" faults, the "stuck-open" fault also needs to be tested. The p-channel transistor with input A is stuck open. When the input AB is set to logic values 10, the output will have logic 1, since the p-channel network connects the VDD to the output and the n-channel network disconnects the output from GND. In this case the value of output in faulty and fault free circuits is the same. When the input AB=01, the n-channel network disconnects the output from GND, and the "stuck-open" fault in the p-channel transistor with input A disconnects the output from VDD. As a result the output of the circuit assumes a high-impedance "Z" state and retains its previous



A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

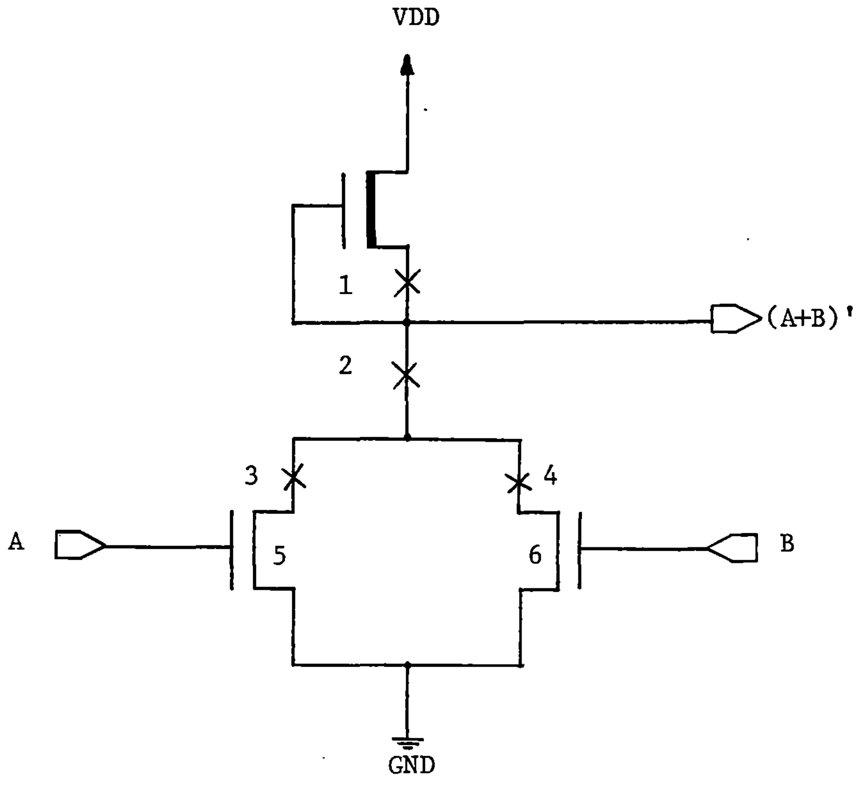


Figure 13. A 2-Input "NOR" Gate Logic Symbol, Logic Function, and Circuit Diagram

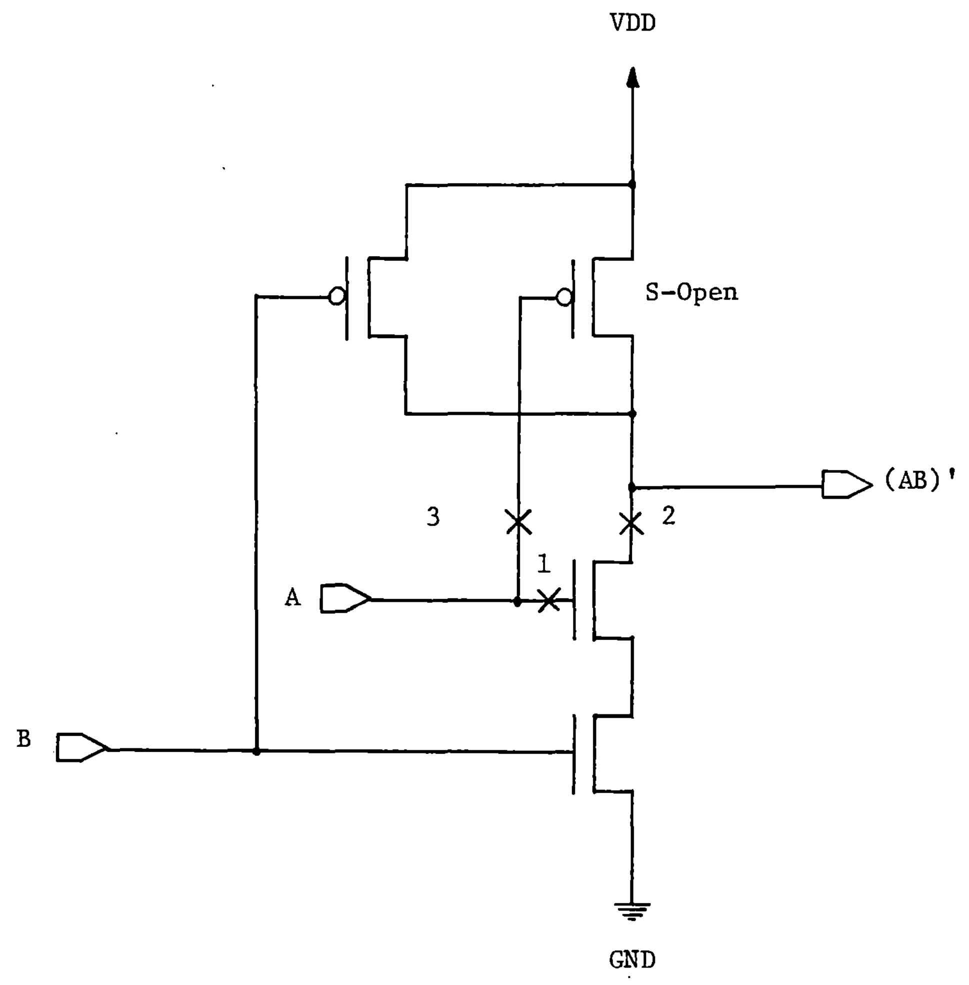


Figure 14. A 2-Input CMOS "NAND" Gate Circuit Diagram

value. If the previous value happened to be logic "1", then the "stuck-open" fault will not be detected. Otherwise, if the previous output value were "0" then the faulty output and fault-free output would differ and the fault would be detected. The test set for testing the CMOS NAND gate of Figure 14 along with the "stuck-at" fault with input AB is 11 01 11 10 and should be applied in that sequence. As was shown before, the test set for testing the 2-input "NAND" gate at the logic gate level is 10 01 11 which is different from the corresponding test set of CMOS "NAND" circuit.

Figure 15 shows a 2-input CMOS "NOR" gate circuit diagram at the transistor level. This circuit has been studied by Wadsack [6]. It has been shown that besides the "stuck-at" fault, the "stuck-open" fault also needs to be tested.

The set for testing the 2-input CMOS "NOR" circuit is 00 10 00 01 and should be applied in that sequence. Again the test set generated at the logic gate level of the 2-input "NOR" gate is 00 10 01 which is different from the corresponding test set of CMOS "NOR" circuits.

In some NMOS circuits the "stuck-open" fault also needs to be tested along with "stuck-at" faults.

Pass Transistor Logic

The next simple circuit is pass transistor logic which does not have an equivalent element at the logic gate level. Pass transistors can be either positive or negative as explained in CSA switching theory in Chapter I. Figure 16 shows both kinds of pass transistors. In Figure 16, A is assumed to be the input, K the control, and B, the output of the pass transistor. In fault free circuits, when K is set to

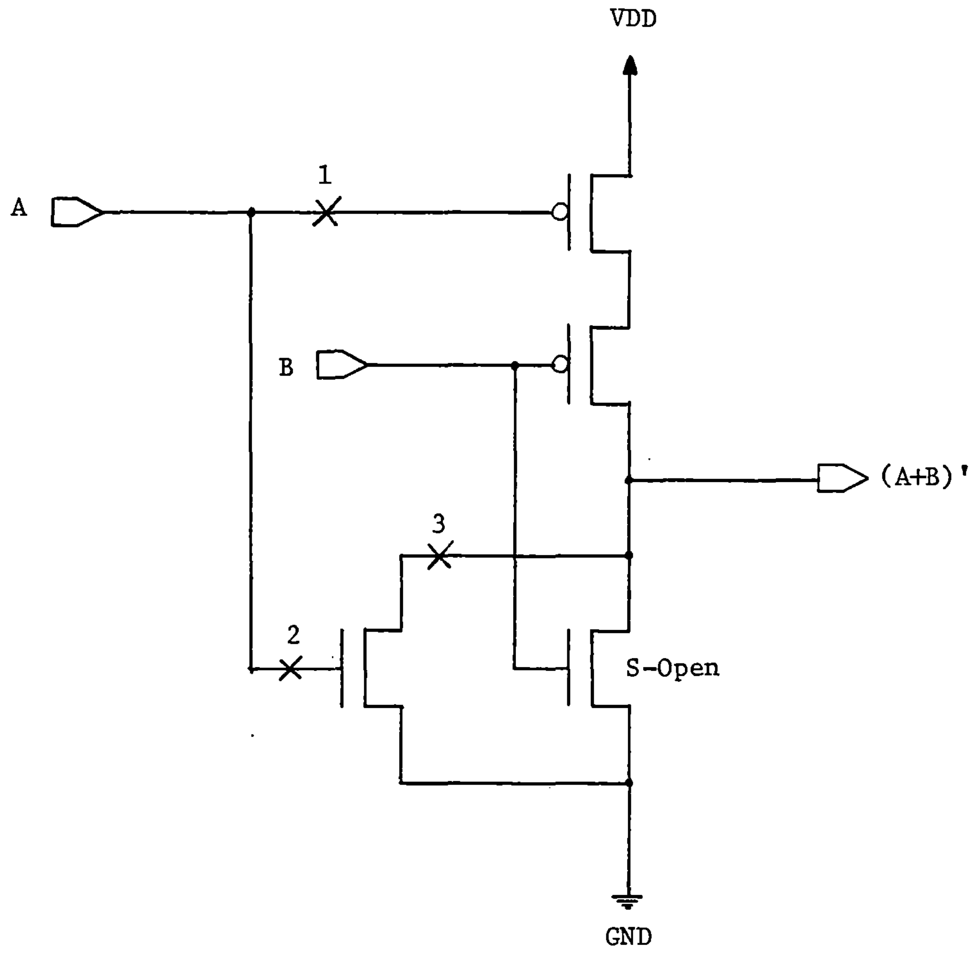
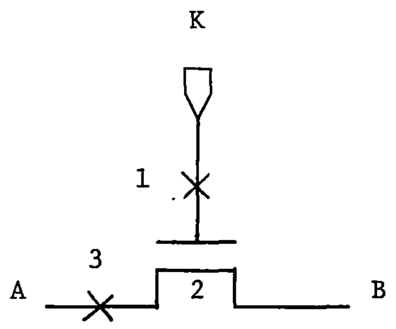


Figure 15. A 2-Input CMOS "NOR" Gate Circuit Diagram

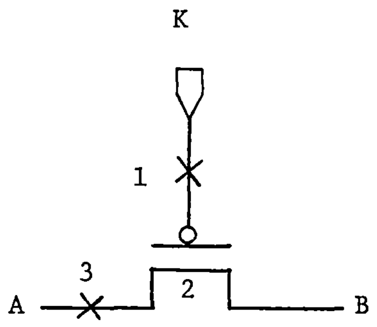
logic value "1" for nMOS transistors ("0" for pMOS transistors) B has the same logic value as A; and when K is set to logic value "0" for nMOS transistors ("1" for pMOS transistors), there is no connection between A and B and a change in A has no effect on B. At this state B retains its previous value.

The pass transistor needs to be tested for logic value "0" and "1" at point A when control "K" is set to logic value "1" in the n-channel ("0" in p-channel) transistor. When the control "K" is set to logic value "0" in the n-channel ("1" in p-channel) transistor, point A needs to be tested for the logic value opposite of the logic value at point B. The total test for pass transistors, to detect all faults for nMOS transistor, is AK=11, 01, and 10 and for pMOS transistors is AK=10, 00, 11 in that sequence. The test set can also be 01 11 00 and 00 10 01 for the nMOS and pMOS transistor, respectively.

A series of pass transistors in a row is called a pass transistor chain. Consider the example in Figure 17(a) which is a chain of two n-channel pass transistors. The test set that can detect all faults for this circuit is AKS=111, 011, 101, and 110 which is exactly like the test set of the 3-input "AND" or "NAND" gate with inputs A, K, and S. Figure 17(b) is a chain of two p-channel pass transistors. The test set that can detect all faults for this circuit is AKS=000, 100, 010, and 001 which is exactly like the test set of the 3-input "OR" or "NOR" gate with inputs A, K, and S. It should be noted that the output B is not equivalent to the outputs of the gates in the above discussion. Another possible test for Figure 17(a) is AKS=011 111 001 010 which is not the same as the test for the 3-input "AND" or "NAND" gate; another test for Figure 17(b) is AKS=100 000 110 101 which is also not the same as the

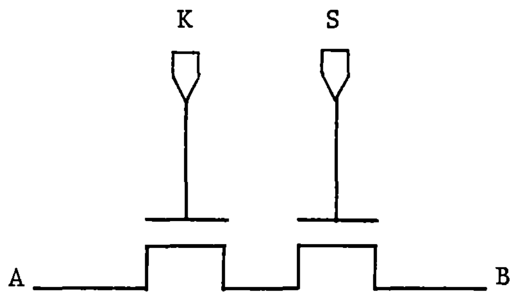


(a) nMOS Transistor

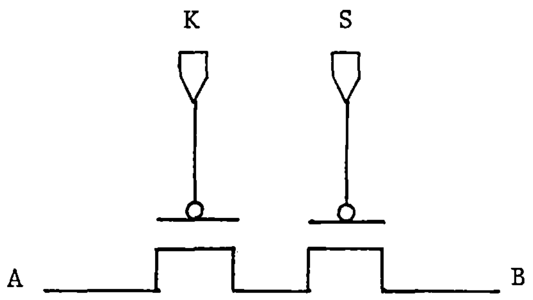


(b) pMOS Transistor

Figure 16. Transmission Gate



(a) Positive Transistors



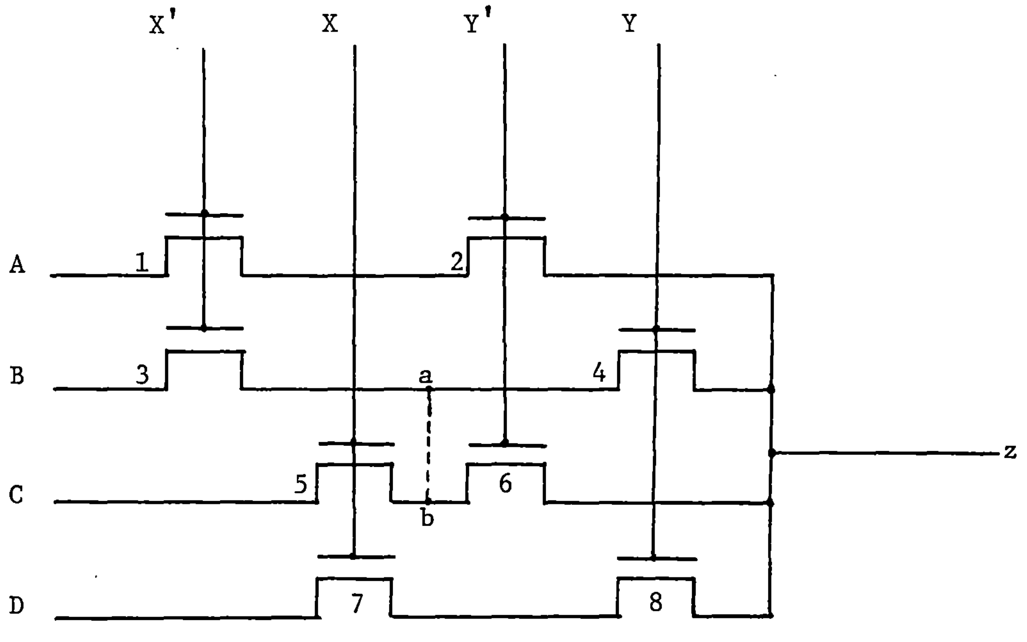
(b) Negative Transistors

Figure 17. Pass Transistor Chain

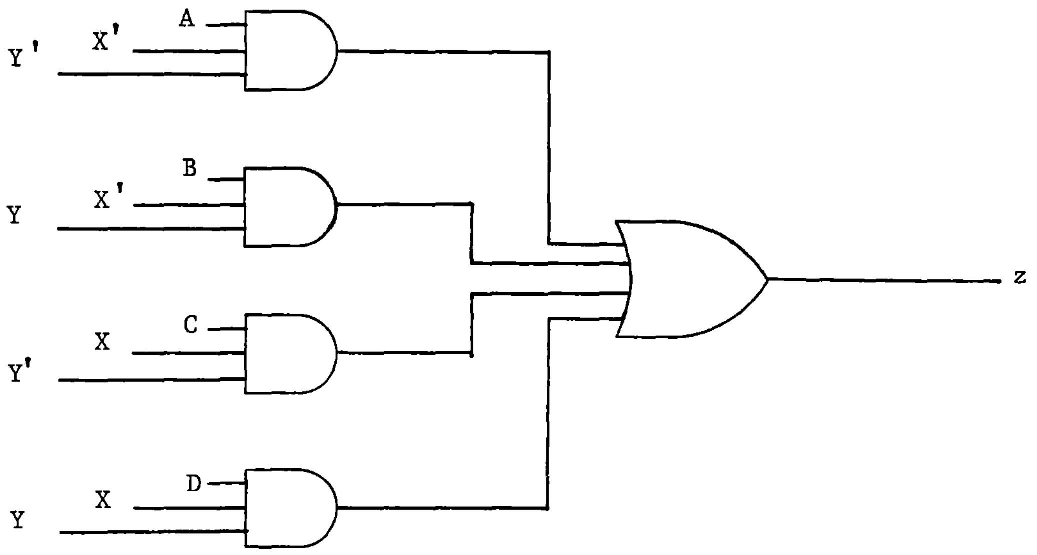
test for the 3-input "OR" or "NOR" gate.

An example of a circuit using a pass transistor chain is the Selector Logic Circuit (SLG) shown in Figure 18(a). Figure 18(b) is the corresponding logic gate level. The generated tests at the logic gate level for "stuck-at" fault using the path sensitization method for SLG is shown in Table I. This test set also detects all "stuck-at" faults at the transistor level of this circuit. As was explained before, four tests are needed to detect all the faults in a chain with two pass transistors. As a result, the total test needed for testing four to one SLG seems to be sixteen (four rows of pass transistor chains with four test per row); but the set of tests that detects the fault(s) in one chain also detects the fault(s) in other rows. For example, test 1 and 5 in Table I detects "stuck-open" fault in transistor 1 and 2 (in row 1 with input A); Test 1 also detects a "stuck-short" fault in transistor 4 and 5 (in row 2 and 3 with inputs B and C, respectively). It should be noted that there are other possible test sets that can detect faults in SLG of Figure 18(a); but the test set of Table I is preferred, since it also detects the "stuck-at" fault at the corresponding logic gate level (Figure 18(b)).

In the circuit 18(a) other possible failures are short faults which may not be detected by "stuck-at" generated tests; thus, additional tests may be needed. Some of these tests may be generated by substituting "don't-care" with a proper logic value ("0" or "1") in the "stuck-at" fault generated tests. Consider a short between point "a" and "b" which cannot be detected by the test set in Table I. This fault can be detected by applying the pattern h) 10XX00 or 1) XX0111 to the inputs A, B, C, D, X, and Y, and check the output "z" for logic value



(a) Circuit Diagram



(b) Logic Diagram

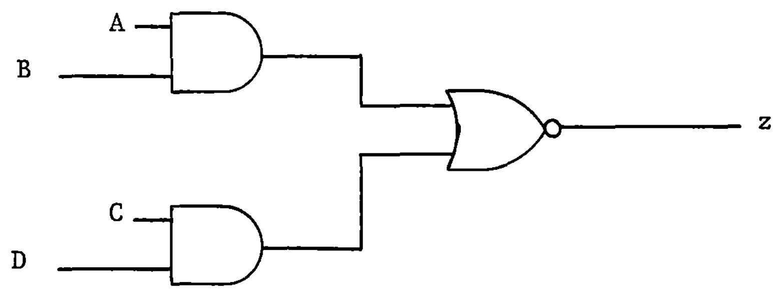
Figure 18. Selector Logic Circuit (SLG)

"0." The test "h" can be generated by replacing logic value "X" with "0" for input B in test 5 of Table I. The test "l" can be generated by replacing logic value "X" with "0" for input C in test 8 of Table I. It is important for one to know that it is quite difficult and sometimes impossible to test for all interconnection shorts. However, it is possible for the designer to use the transistor level implementation of the MOS circuit layouts to select possible fault locations. In that case, it is proper to say that generating tests for "short-faults" is layout dependent.

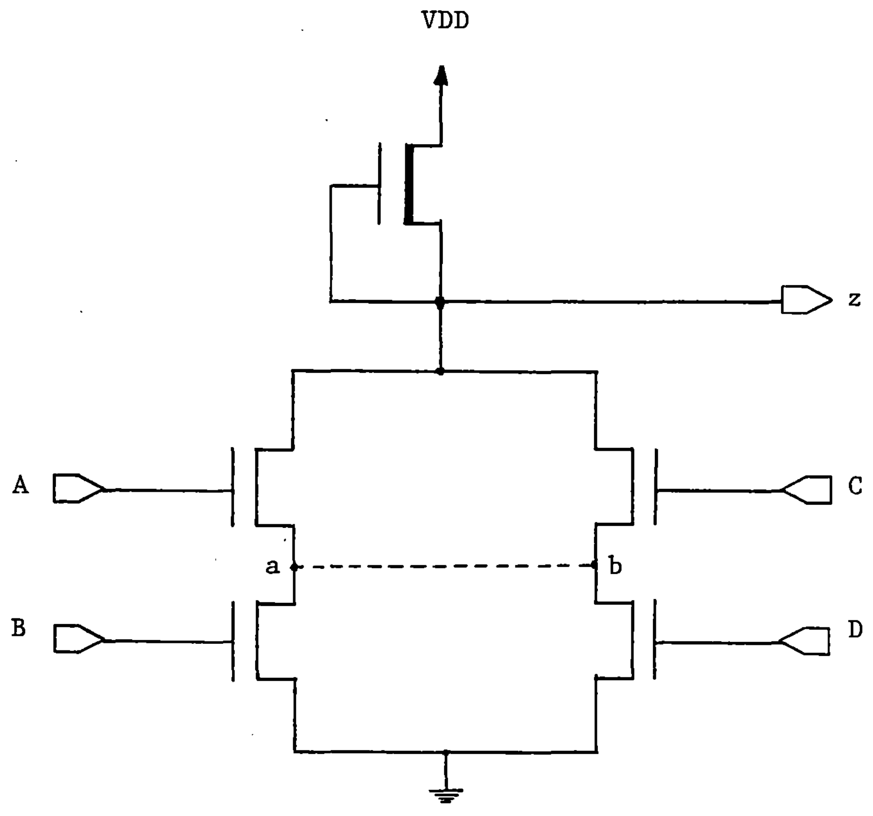
Complex Cell

Figure 19 shows a complex cell circuit diagram and equivalent logic symbol. The test set generated at the logic level of this circuit using the "stuck-at" fault model is shown in Table II. This set of tests also detects all "stuck-at" faults at the circuit or transistor level (Figure 19(b)). Additional tests may be needed to detect all of the possible short faults. For example, the short between point a and b shown in Figure 19(b) is detected by applying the pattern 0110 to the inputs A, B, C, and D which is not one of the test patterns in Table II.

Other advantages of generating tests for MOS circuits at the transistor level, rather than logic gate level, is that it is possible to design a significantly smaller test set for some particular circuits. This is possible by taking advantage of structural properties of MOS circuits. These include a pass transistor which is used as a part of realizing multiple paths in a particular MOS circuit and must be replaced with multiple gate at the logical gate level to realize the same functions. Such a circuit is shown in Figure 20. Figure 20(a)



(a) Logic Diagram



(b) Circuit Diagram

Figure 19. Complex Circuit

TABLE I

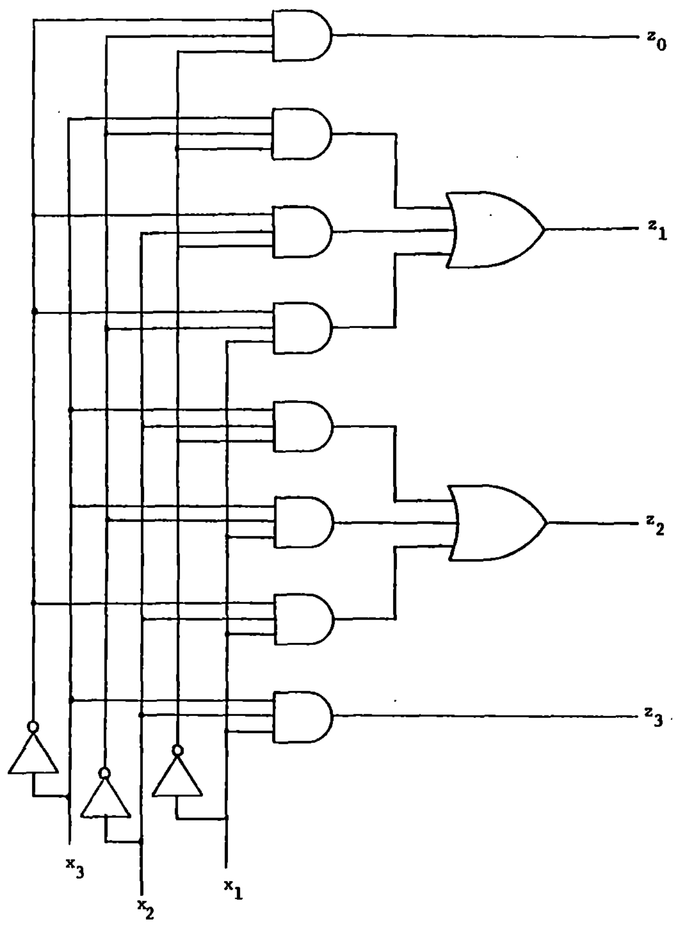
TEST SET FOR SLG CIRCUIT USING PATH SENSITIZATION

Test No.	Inputs A	B	C	D	X	Y	Output Z
1	0	1	1	X	0	0	0
2	1	0	X	1	0	1	0
3	1	X	0	1	1	0	0
4	X	1	1	0	1	1	0
5	1	X	X	X	0	0	1
6	X	1	X	X	0	1	1
7	X	X	1	X	1	0	1
8	X	X	X	1	1	1	1

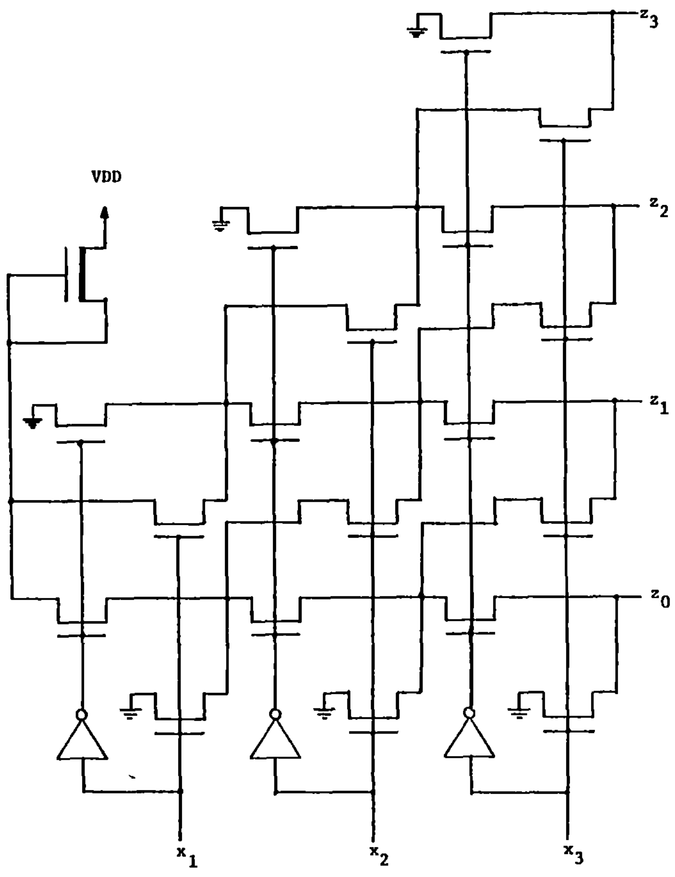
TABLE II

TEST SET FOR COMPLEX CIRCUIT OF FIGURE 19(a) USING PATH SENSITIZATION

Test No.	Inputs A	B	C	D	Output Z
1	0	1	0	1	1
2	1	0	1	0	1
3	1	1	0	X	0
4	0	X	1	1	0



(a)



(b)

Figure 20. A 3-Input Gate Level Tally Circuit (a), The Corresponding Transistor Level Network (b)

TABLE III

TEST SET FOR 3-INPUT TALLY CIRCUIT AT THE GATE LEVEL USING PATH SENSITIZATION

Inputs			Outputs			
x_1	x_2	x_3	z_0	z_1	z_2	z_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	0	0
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

TABLE IV

TEST SET FOR 3-INPUT TALLY CIRCUIT AT THE TRANSISTOR LEVEL

Inputs			Outputs			
x_1	x_2	x_3	z_0	z_1	z_2	z_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

shows the gate level description of a tally circuit while 20(b) is the equivalent circuit level representation. The logic gate level description of a symmetric function with n inputs requires 2**n "AND" gates along with n-1 "or" gates to realize the function. Since output of every "AND" gate needs to be tested for s-a-0 (see Figure 20(a)), and there is only 2**n possible combination inputs, the circuit should be exhaustively tested for all 2**n possible inputs at the logic level. However, only 2n tests are needed to detect all faults using transistor level implementation of a symmetric function [26]. Table III shows all possible tests for n=3 for a tally circuit and Table IV shows the actual tests needed to test that circuit.

Summary

This chapter has shown the advantage of generating tests for a particular circuit at the transistor level rather than the gate level. In some cases, by using the structural properties of the MOS circuits, it is possible to design much smaller test sets at the transistor level rather than gate level. Short circuit faults can be detected much more easily at the transistor level. But the most important result is that, in some MOS circuits, "non-stuck-at" faults ("stuck-open" fault) cannot be detected using a test set generated at the logical gate level.

CHAPTER IV

WIRED-GATES-SWITCHES NETWORK

Introduction

CSA switching theory implements the structural properties of MOS design at the circuit level. Therefore, a method which generates tests to detect faults occuring in the CSA switching model, will also detect the same faults occuring at the corresponding transistor level of that circuit. The switches (positive and negative) in the CSA switching theory are bidirectional, which results in bidirectional lines throughout the network. Therefore, a wired-logic and a fanout node in CSA switching theory cannot simply be recognized. Because of these bidirectional lines (or switches) in the CSA structure, it is quite difficult to follow the direction of a signal when generating a test. That is, a path which begins with a particular primitive input may end at the same input (looping) or other primitive inputs rather than a primitive output. Thus, in some CSA networks, a lot of unnecessary process might be taken before a reasonable path is generated. This would cause loss of time during test generation process. Therefore, a new model is needed.

The "wired-Gates-Switches" (wGS) network, which is the modified version of CSA switching theory, is introduced in this chapter. The wGS also implements the structural properties of MOS digital circuits for testing purposes. The wired-logics in CSA networks are replaced with one or more 2-input "wired-Gate" elements to implement wired logic functions and also specify the direction of signals throughout the networks. The direction of the signal for every line in wGS networks is defined; for every primitive input in the networks, there is at least one path to a primitive output. The path sensitization, the method for generating tests at the logical gate level, can be used to generate tests at the transistor level of MOS digital circuits, using a wGS network representation of the circuits.

Elements in wGS Networks

The elements in wGS networks are: 1) three-terminal, single direction, positive switches which correspond to the n-channel Fet or nMOS transistors with defined direction; 2) three-terminal, single direction, negative switches which correspond to the p-channel Fet or pMOS transistors with defined direction; 3) 2-input "wired-Gate" devices which implement the wired logic operation in MOS circuits for a unidirectional signal; 4) 5-terminal device bidirectional positive switches correspond to the n-channel Fet or nMOS transistors. The 5-terminal device has two inputs, two outputs, and a key (control) which turns the switch "ON" when it is set to logic value "1" or "1^w"[1] and "OFF" when the key is set to logic value "0^w." In this device, one set of input and output lines realize one direction and the other input and output lines realize the other direction in the switch. It is important to know that even though the switch is bidirectional, only one direction is allowed to be used at a time for a particular signal from

[1] The symbol "w" represents the "weak of".

input to the output of the circuit through this element. Lastly, the wGS network includes 5-terminal device bidirectional negative switches corresponding to the p-channel Fet or nMOS transistors. Each of these devices has two inputs, two outputs, and a key (control) which turns the switch "ON" when it is set to logic value "0" or "0^w" and "OFF" when the key is set to the logic value "1" or "1^w." In this device one input and one output lines realize one direction and the other input output lines realize the other direction in the switch. Again, it is important to know that although the switch is bidirectional, only one direction is allowed to be used at a time for a particular signal from input to the output of the circuit through this element.

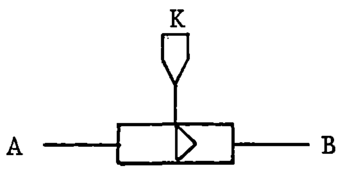
Switches

Switches in wGS networks are the basic components of any switching circuit. The positive and negative single direction switches and their logic functions (primitive functions) are shown in Figure 21. The single direction positive (negative) switches in wGS networks are equivalent to positive (negative) switches in the CSA switching model explained in Chapter II when they are used to realize a single direction signal. The control terminal K is the same, but the data terminals D_1 and D_2 change to the input data terminal A and output data terminal B; A and B are not symmetric data terminals. When S is "ON" (K has logic "1" for positive switches and logic "0" for negative switches), A and B are connected together and the output of S (terminal B) has the same logic value as A. When S if "OFF" (K is logic "0" for positive and logic "1" for negative switches), A and B are disconnected and the output of S (terminal B) is said to have a logic value "Z" (high impedance); B retains its previous logic value.

The bidirectional positive (negative) switches in wGS networks are equivalent to positive (negative) switches in the CSA switching model when they are used to realize bidirectional signals. The positive and negative bidirectional switches in CSA and wGS networks are shown in Figure 22. The bidirectional switch S is changed by control terminal K exactly like single direction switches. In switch S either terminals A and B or C and D along with terminal K are used for a particular signal depending upon the direction of the signal through the switch. D_1 and D_2 in switch T are equivalent to A and B in switch S, respectively, when a signal travels from D_1 to D_2; D_1 and D_2 are equivalent to C and D, respectively, when a signal travels from D_2 to D_1. In other words, $A \rightarrow B$ is equivalent to $D_1 \rightarrow D_2$ and $C \rightarrow D$ is equivalent to $D_2 \rightarrow D_1$ in switch S and T. Although the switch S used in the wGS networks handles bidirectional signals, the individual data terminals A, B, C, and D are not bidirectional.

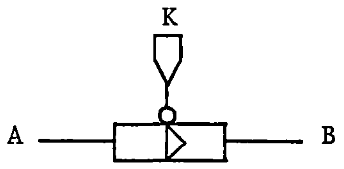
Wired-Gates

Wired-Gates in the wGS networks play a central role in logic behavior of the circuit model. Figure 23 shows a wired-Gate, its logic behavior in terms of logic value V6 = $(0,1,1^w,0^w,U,Z)$, and its primitive functions in terms of logic value V5 = $(0,1,1^w,0^w,Z)$. The primitive functions, Part (c), are the response of a wired-Gate in the non faulty circuits which will be explained in detail for generating tests at the wGS networks in Chapter V. In this research, every wired-Gate device will be determined by its output terminal. The terminals A and B in the wired-Gate G are symmetric. The wired-Gate G



A	K	B
0	0	Z
0	1	0
1	0	Z
1	1	1

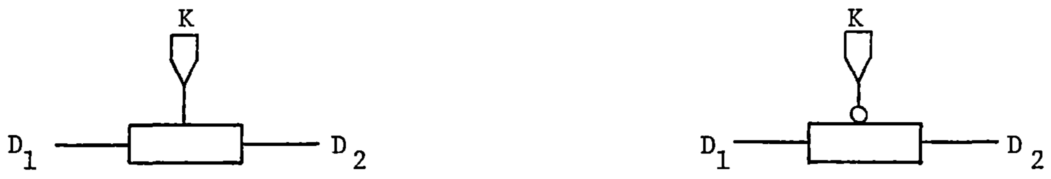
(a) Positive Switch "S"



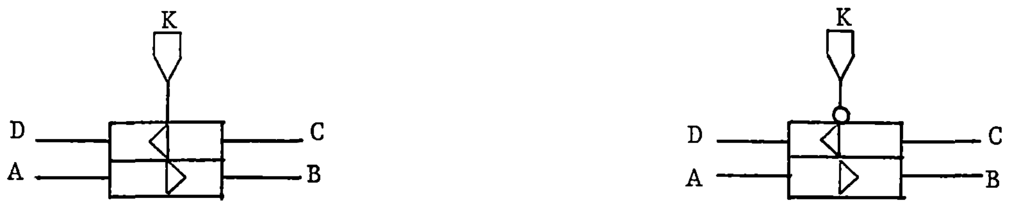
A	K	B
0	0	0
0	1	Z
1	0	1
1	1	Z

(b) Negative Switch "S"

Figure 21. Single Direction Switches in the wGS Networks and their Logic Functions (Primitive Functions)

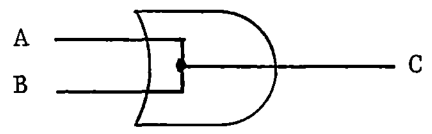


(a) In CSA Networks



(b) In wGS Networks

Figure 22. The Bidirectional Positive and Negative Switches



(a) Logic Symbol

A	B	C
0	$0,0^w,Z$	0
0	1^w	0
0	1	U
0^w	1^w	U
1	$1,1^w,Z$	1
1	0^w	1
0	$0^w,Z$	0^w
1^w	$1^w,Z$	1^w
Z	Z	Z
U	d	U

(b) The behavior with respect to $V6=(0,1,0^w,1^w,U,Z)$; d denotes "don't care" value.

Inputs			Output
A		B	C
Allowed	Selected		
$(1^w,0)$	Z	0	0
------	Z	0^w	0
$(0^w,1)$	Z	1	1
-------	Z	1^w	1
-------	Z	Z	Z

(c) Primitive function.

Figure 23. The "wired-Gate" Device.

realizes the same logic function as connector C explained in Chapter II, Figure 7. The logic operation of a "wired-Gate", like a connector, is denoted by # (i.e. for Figure 23 output G = #(A,B)).

Figure 24 is a realization of wired-logic in terms of "wired-Gate" devices when more than two connectors are joined together. Part A shows a wired-logic with four input lines, X_1, X_2, X_3 and X_4. This can be implemented as three connectors C_1, C_2, and C_3 in the CSA networks shown in Figure 24(b) or 24(c). The equivalent "wired-Gate" implementation of Figures 24(b) and 24(c) are shown in part (e) and (d) of same figure, respectively. The logic operation of Figure 24(e) with three gates G_1, G_2, and G_3 is:

$$Z = \#(G_3)=\#[\#(G_2),X_4]=\#(\#[\#(G_1),X_3],X_4)$$
$$= \#(\#[\#(X_1,X_2),X_3],X_4) = \#(X_1,X_2,X_3,X_4)$$

and the logic operation of Figure 23(d) with three gates G_1, G_2, and G_3 is:

$$Z = \#(G_3)=\#[\#(G_1),\#(G_2)]$$
$$= \#[\#(X_1,X_2),\#(X_3,X_4)] = \#(X_1,X_2,X_3,X_4)$$

See the definition (1) and (2) in Chapter II.

In both Figures 24(d) and 24(e), the three "wired-Gates" G_1, G_2, and G_3 are designed to carry the same physical properties as connector C in Figure 24(a). For the same reason part (a) and (b) of Figure 25 representing the same structure for testing purposes, since C_1 and C_2 are practically shorted together.

In the case when the output of wired-logic has more than one fan out, the wired-logic node is replaced with one or more "wired-Gate" devices and a fanout node. An example of this case is shown in Figure 25. Part (b) is connector C implementing a wired-logic operation in CSA

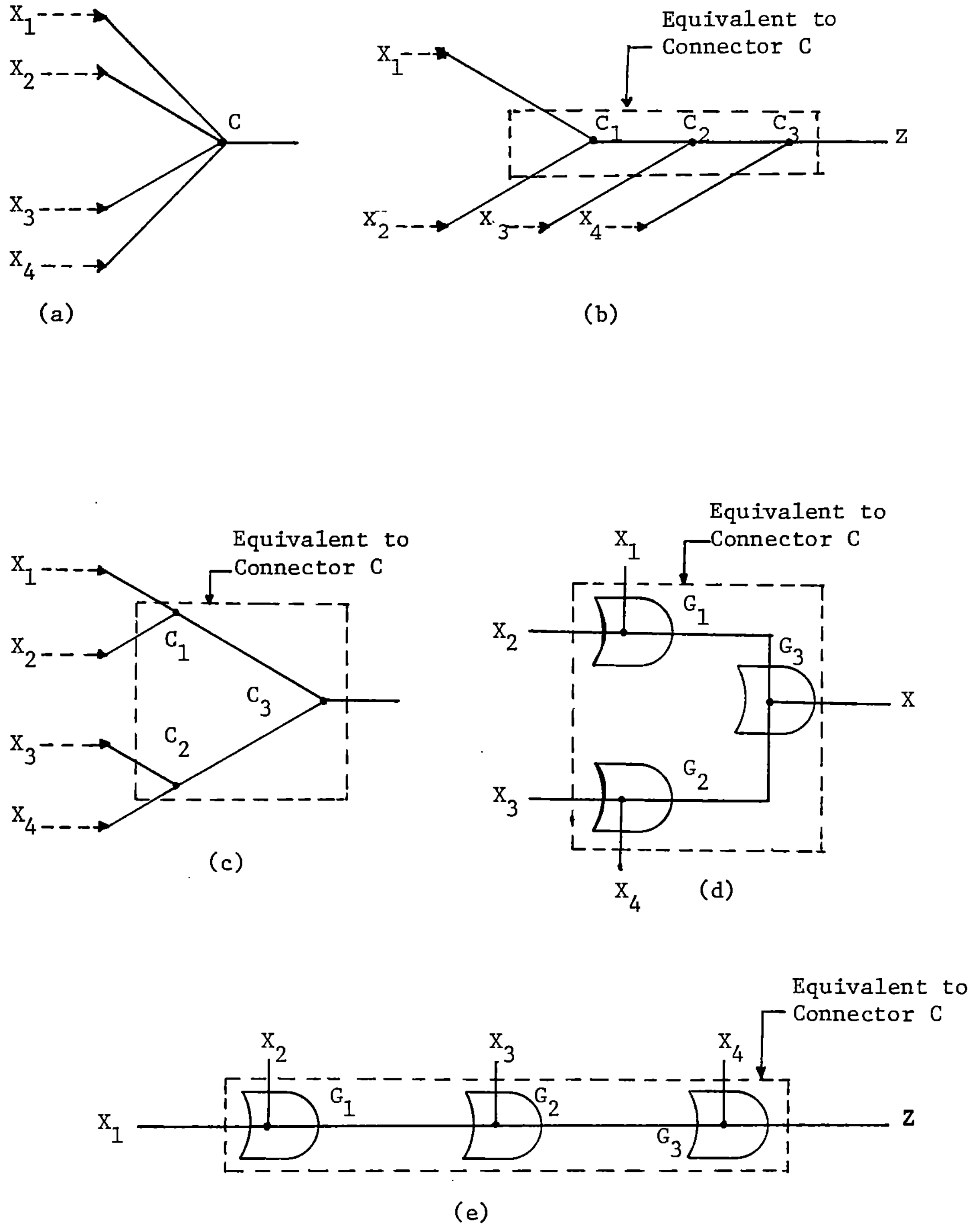
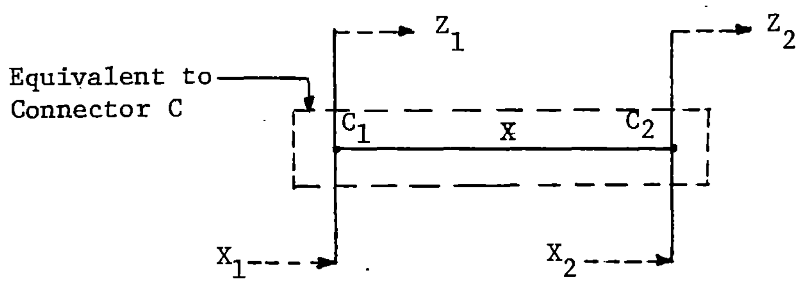
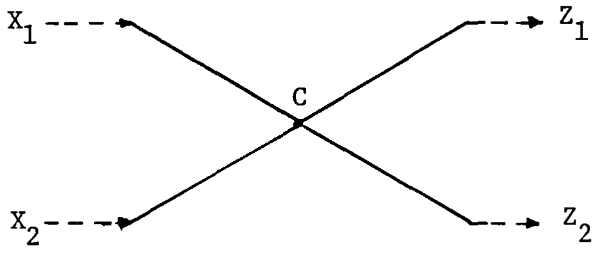


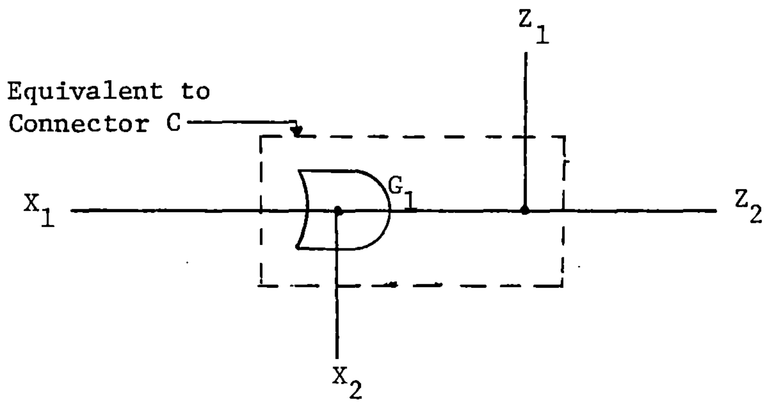
Figure 24. Implementation of Wired-Logic in terms of "wired-Gate" Devices



(a) In CSA Networks



(b) In CSA Networks



(c) In wGS Networks

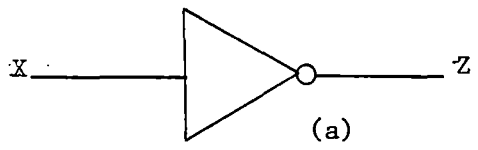
Figure 25. Wired-Logic Representation with Two Fanouts and Two Inputs

switching theory with two inputs X_1, and X_2 and two fanouts Z_1 and Z_2. Part (c) is the equivalent representation in wGS networks. Again G_1, and the fanout node in Figure 25 (c) carry the same physical properties as connector C.

Wired-Gates-Switches Networks

The wired-Gates-Switches networks are combinations of "Switches" (single and bidirectional positive and negative), wired-Gates, and lines which interconnect switches and wired-Gates. The wGS networks implement the transistor level of MOS digital integrated circuits for test generation purposes. The nMOS and pMOS transistors are replaced with positive and negative switches (both single and bi); the wired-logic nodes are replaced with one or more wired-Gates; and pull-up (pull-down) devices (attenuator in CSA switching theory, transistors or resistor at the circuit level) are replaced with lines carrying logic value "1^W" (0^W). Figure 26 shows a very simple (an inverter) logic symbol, circuit diagram, logic function, equivalent CSA network, and equivalent wGS network.

It is very important for one to know that the connectors in CSA networks cannot always be replaced by "wired-Gate" elements. Some of the transistors in the MOS circuits, specifically sequential and two-phase clocks circuits, which have been replaced with switches in CSA networks are bidirectional and single direction cannot be assumed through lines going through these switches. In these type circuits, connectors may sometimes act like fanouts and/or the signal may change direction to or from connectors depending upon selected input signals. In this case, a five terminal bidirectional device replaces the switch;



x	z
0	1
1	0

(b)

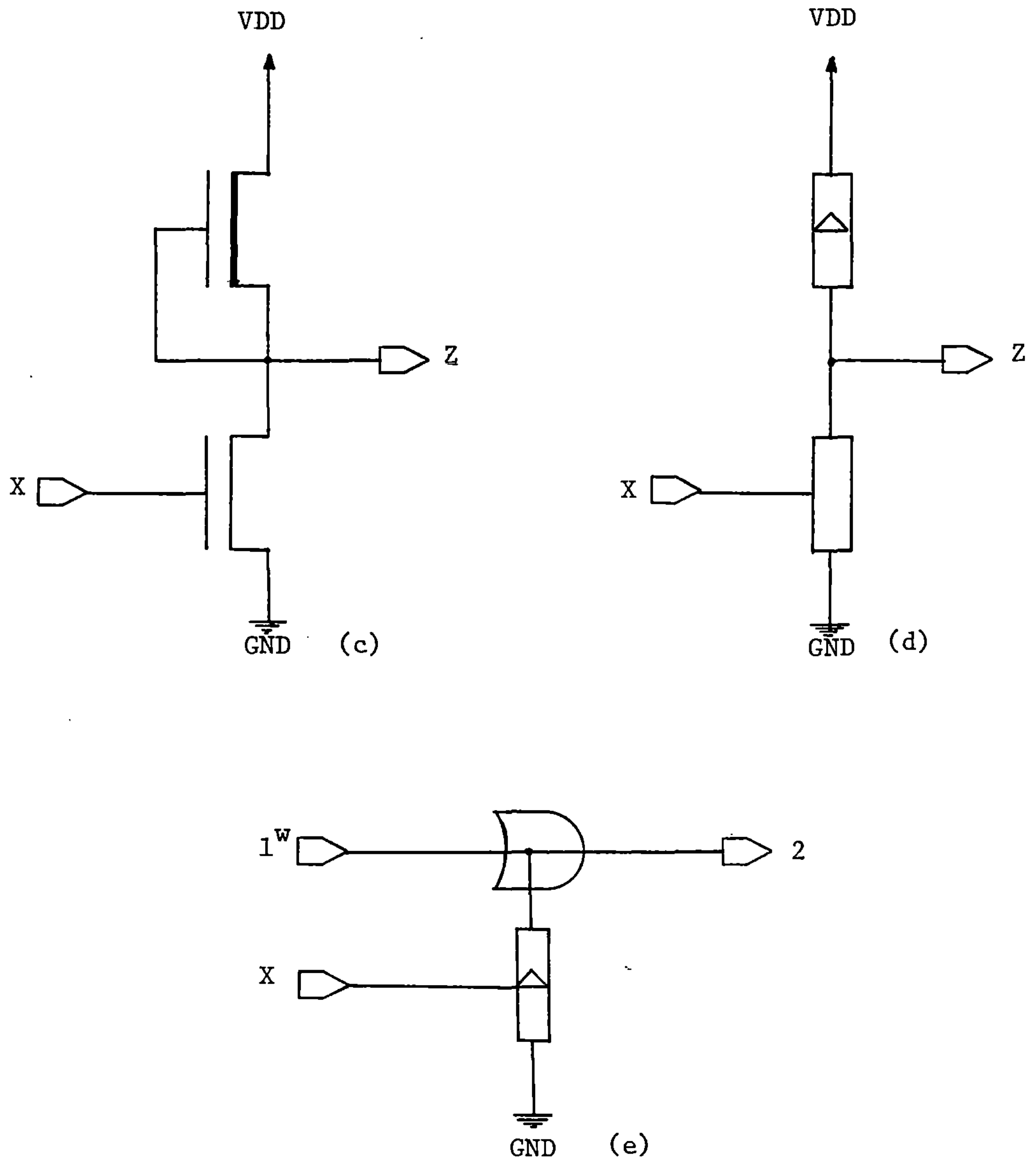


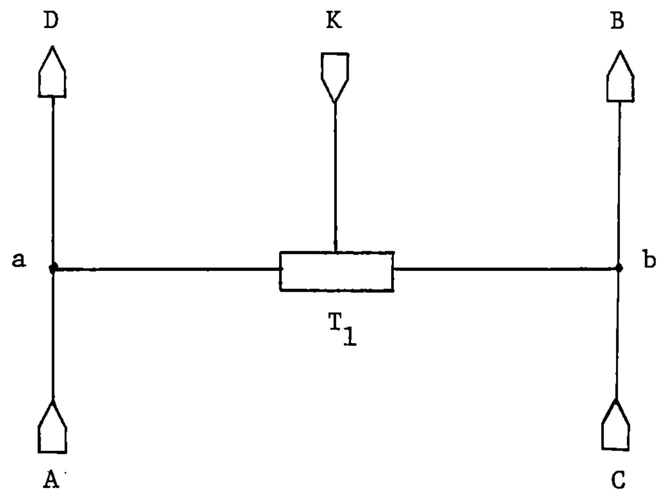
Figure 26. Inverter

and those particular connectors are replaced with one or more "wired-Gate" elements and fanout nodes to realize the logic function of the connectors required by the circuit.

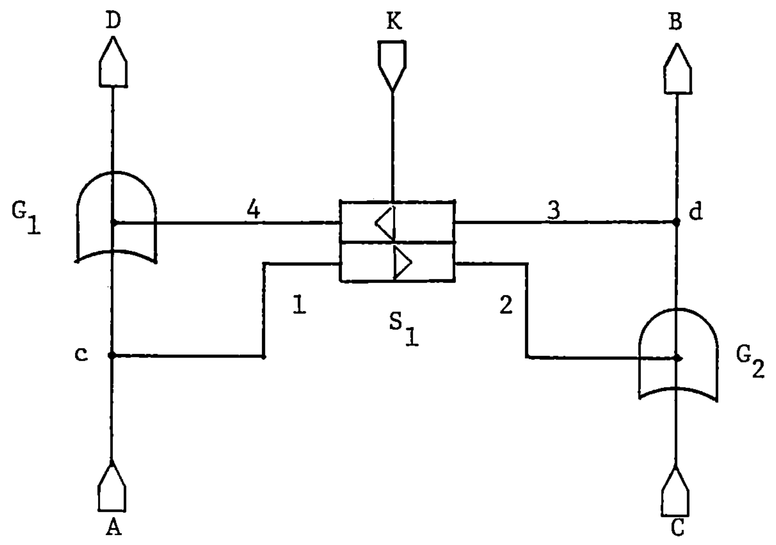
An example of such a connection is the bidirectional positive switch "T_1" in Figure 27(a), which is replaced by a five terminal positive switch "S_1" with two inputs 1 and 3, and two outputs 2 and 4. As was stated before, in this switch either the input 1 and the output 2 or the input 3 and the output 4 can be used at the same time for a single test. Thus, in Figure 27(b) the signal travels from A to B through the input 1 and the output 2, and travels through the input 3 and the output 4 from C to D. Connector nodes a and b in the CSA representation, cannot be replaced with a single "wired-Gate", since a single output for these nodes cannot be identified. However, each can be replaced with a combination of a "wired-Gate" and a fanout node. The "wired-Gate" G_1 (G_2) and fanout node c(d) in Figure 27(b) implement the connector a(b) in part (a) of the same figure.

Theorem: Assume that a MOS circuit implements a particular logic function without a transistor used for a bidirectional switch. Then besides primitive input(s) and output(s) (if used as I/O lines), either there is no bidirectional line(s) in the circuit, or each bidirectional line is between two Wired Logic Nodes (WLN) (i.e. the line "X" between C_1 and C_2 in Figure 25(a)) where both WLN and the line can be replaced with one WLN. (i.e. WLN C in Figure 25(b))

Proof: There is no bidirectional switch in the circuit and possible attenuator in the circuit is also single direction; thus, any line to or from every switch and/or attenuator is single direction. This proves that a bidirectional line which is not a primitive input or



(a) In CSA Networks

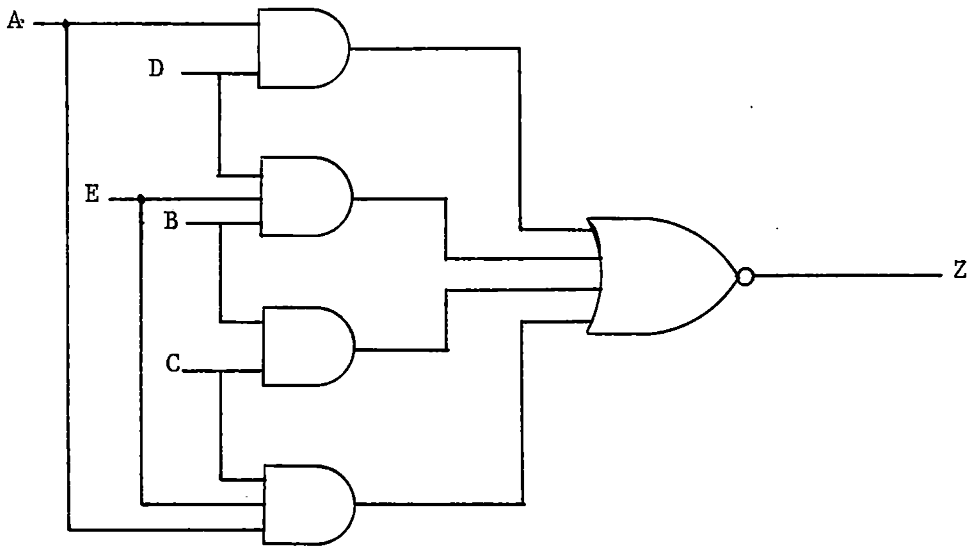


(b) In wGS Networks

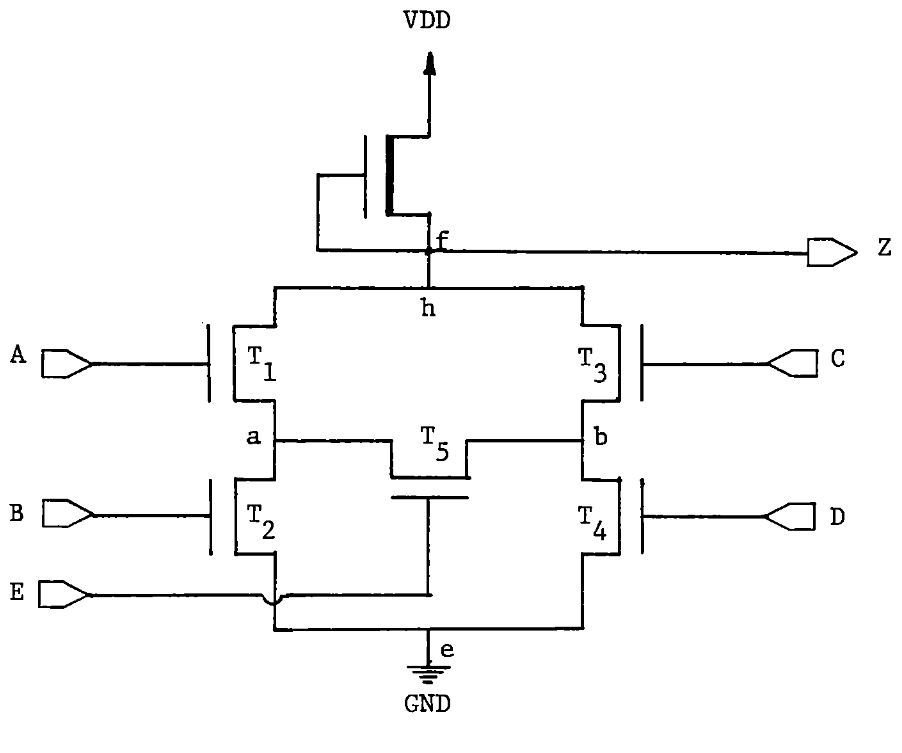
Figure 27. The Implementation of Bidirectional Signal in the wGS Networks

output must be between two WLN. Assume a bidirectional line "X" connecting two connector C_1 and C_2 shown in Figure 25(a). Since "X" is only a bidirectional line and no other element is between C_1 and C_2, WLN C_1 and C_2 are practically shorted together. Therefore, nodes C_1 and C_2 can be replaced by a single node C. In that case, the node C carries all physical properties of connector C_1, C_2, and line "X".

An example of a circuit using a bidirectional switch is the bridge circuit. Figure 28 shows the logic gate level and transistor level (circuit level) bridge circuit. The wGS network representation of the bridge circuit of Figure 28 is shown in Figure 29. The bidirectional positive switch S_5 implements the nMOS transistor T_5 shown in Figure 28(b). The "wired-gate" $G_1(G_2)$ and fanout node c(d) implement the connector a(b) in Figure 28(b). The wired-logic nodes h and f at the circuit level of the bridge circuit are replaced with wired-Gate G_3 and G_4, respectively. As it is shown in Figure 29, the pull-up depletion node nMOS transistor at the circuit level of the bridge circuit is replaced with a line carrying the logic value "1^w" (read as weak one). There are no bidirectional lines in wGS networks, thus, in this network and other wGS networks there is at least one path from every primitive input to the primitive output; VDD and GND are also considered to be primitive inputs. Since every path which starts from a primitive input will end at a primitive output, and vice versa, the known path sensitization method at the logical gate level of digital circuits with proper fault models can also be applied to wGS networks to generate tests for MOS digital circuits.



(a) Logic Gate Level



(b) Transistor Level

Figure 28. Bridge Circuit

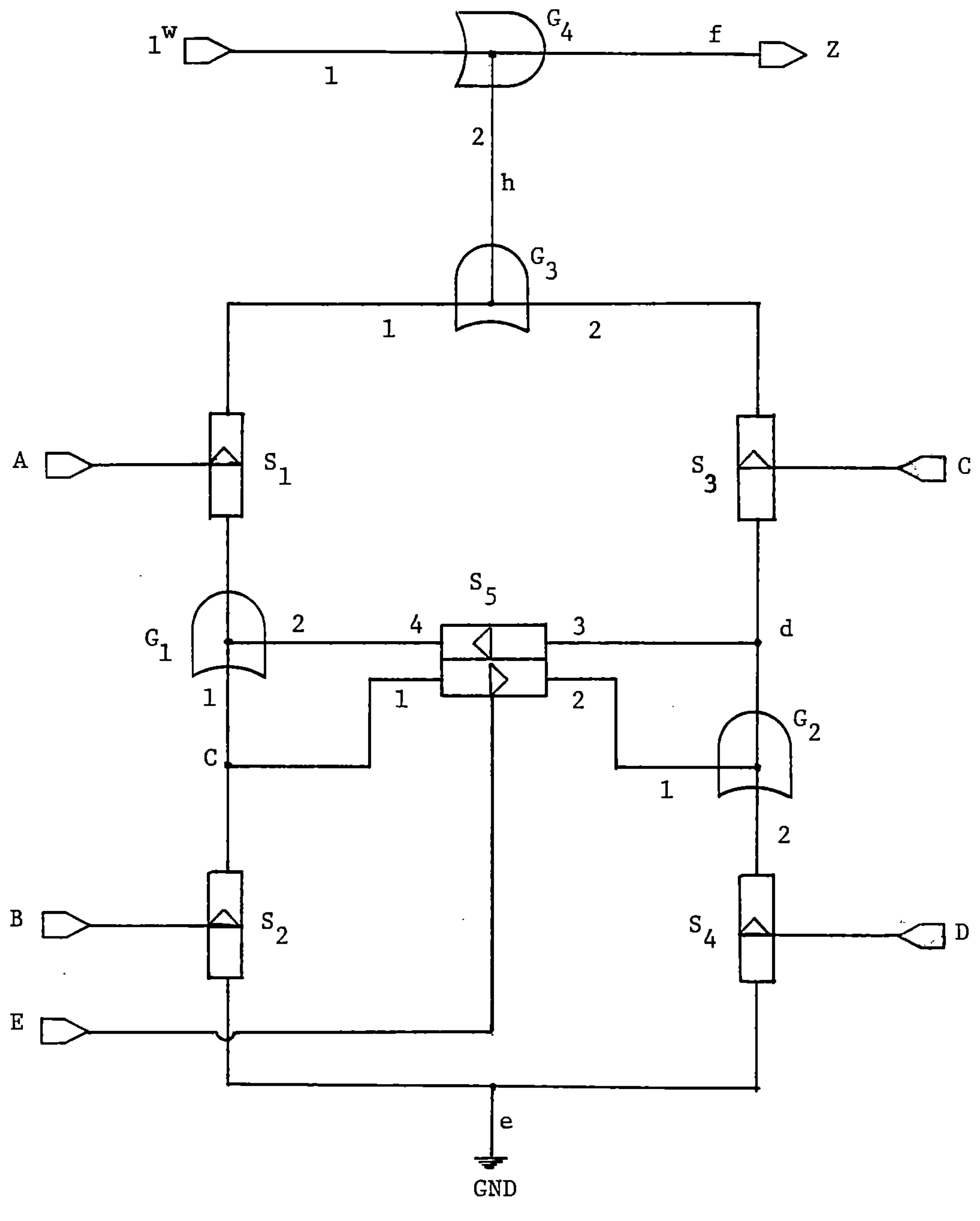


Figure 29. The wGS Network for the Bridge Circuit

Summary

In this chapter, the wGS networks and the modified version of CSA networks for test generation purposes were introduced. Five terminal positive and negative switches along with "wired-Gate" elements and fanout nodes were shown to be useful in implementing bidirectional lines to single direction lines throughout the circuit level of MOS digital design. The wGS networks implement the structural properties of MOS circuits. For every primitive input, there is at least one path to a primitive output and for every primitive output, there is at least one path to a primitive input in wGS networks. Path sensitization, with the proper fault model can be used to generate tests for wGS networks and, therefore, for MOS digital circuits.

CHAPTER V

TEST GENERATION AT THE TRANSISTOR LEVEL FOR MOS COMBINATIONAL LOGIC CIRCUITS

Introduction

In the previous chapters the available methods for generating tests at functional and logical gate levels and the CSA switching theory of MOS technology for digital circuits has been surveyed. It has been shown that the test set generated at the logical gate level is not always adequate for testing MOS design. Also, it has been shown that a test set generated for MOS circuits at the transistor level can be more efficient, for some circuits, than generated tests at the equivalent gate level structure. There is no known method for automatically generating test for digital circuits at the transistor level. The wGS model, the modified version of CSA switching theory introduced in Chapter IV, is designed to implement the structure of MOS circuits for test generation purposes; therefore, a method which generates tests for the wGS networks also generates tests for the equivalent MOS digital circuits.

An extension of existing methods of generating tests at the logical gate level is introduced in this chapter to generate tests for wGS networks. The method is called Critical Path Test Generation at the Transistor Level (CPTGTL), and it is specifically developed for generating tests for MOS combinational circuits. The CPTGTL use stuck-switches (either stuck-open or stuck-closed) faults to generate tests. Besides stuck-at faults, stuck-open faults will also be detected using a test set generated by the CPTGTL. Although the CPTGTL is not designed to detect short circuit faults, one should be able to use this method to generate tests for specific short faults using the transistor level implementation of a particular circuit. However, in most circuits, some of the short circuit faults will be detected automatically using tests generated by the CPTGTL.

A fault model, definitions, and a method of generating tests are explained in this chapter. A PL/1 program is written to demonstrate the CPTGTL algorithm. A selection of examples are used to examine the CPTGTL algorithm and program.

Fault Models

In previous chapters various types of faults in MOS circuits have been discussed, and it has been shown that the generated test set based on a "stuck-at" fault model detects all transistor failures along with some other possible faults in particular MOS circuits. The failures considered for testing a wGS network occur at the switches (transistors). The fault models used are switches faults which are either "stuck-open" or "stuck-short".

Definition 5.1 - A switch "S" in wGS networks shown in Figures 21 and 22 is said to be permanently "stuck-open" (permanently "OFF"), no matter what logic value point K is set to, if logic values at point A(C) stay completely independent of logic value at point B(D).

Definition 5.2 - A switch "S" in wGS networks shown in Figures 21 and 22 is said to be permanently "stuck-short" (permanently "ON"), no matter what logic value point K is set to, if point B (D) always copies the logic value of point A (C).

Definition 5.3 - A positive (negative) switch "S" in a wGS network shown in Figures 21 and 22 is said to be fault free, if both points A(C) and B(D) carry the same logic value when point K is set to logic value "1" ("0"); the logic values of A(C) and B(D) stay independent of each other when point K is set to logic value "0" ("1").

There are other possible faults. A short between lines in a circuit has received less attention in the test generation method introduced in this chapter. However, if it is desired, one can use the same method to generate tests for a particular short-fault using wGS implementation of MOS circuits. This can be done by using a dummy positive or negative switch between the line which is going to be tested for short-circuit, then test this switch for stuck-short fault.

Logic Values

For simplicity, in fault free CMOS wGS networks only three possible logic values, "0", "1", and "Z" are assumed. However, in a faulty CMOS circuit logic value "U" (unknown) is also a possibility, but a test for "U" does not need to be generated. Therefore, the logic behavior of CMOS wGS networks is V3=(0,1,Z). The logic value "U" is also called an intermediate logic value which also has been shown as "I." [26]

In the fault free NMOS wGS networks, with lines carrying "1^W" replacing the pull-up attenuator in CSA networks, a node is denoted by four possible logic values. Logic value "1" corresponds to VDD, input high signal and output high signal. It should be noted that the output-high of pass transistors (switches) is weaker than VDD, but for the purpose of generating tests is assumed to be logic "1". Logic value "0" corresponds to GND, input low signal and output low signal. Again, depending upon the impedance of transistors, the output-low of pass transistors (switches) is weaker than GND; but for testing purposes logic value "0" is sufficient. Logic value "Z", or high impedance, corresponds to the output of opened pass transistor logic in MOS circuits or output of opened switches in wGS networks. Logic value "1^w" is the output of a pull-up replaced by lines. For testing NMOS circuits implemented by wGS networks, it is necessary to distinguish between logic "1^w" and "1". In reality, in a faulty NMOS circuit, a node having logic value "U" is a possibility; but for the purpose of generating tests using wGS networks, logic value "U" does not need to be considered. Thus the logic values of NMOS circuits represented by wGS networks is $V4=(0,1,1^w,Z)$.

For the circuits which use pull-down devices instead of pull-ups throughout the systems, like PMOS circuits, logic behavior is $V4=(0,0^w,1,Z)$. As a result, the logic values of any MOS digital circuit represented by wGS networks for testing purposes is $V5=(0,0^w,1,1^w,Z)$.

Test Generation for wGS Networks

To generate tests for wGS networks, using "stuck-switch" faults, switches (positive and/or negative) must be examined for both "stuck-short" and "stuck-open" faults. Figure 21 shows a positive switch, a negative switch, and corresponding logic functions (truth table). In order to determine the direction of a signal through a switch, the input and output of that switch needs to be defined. In Figure 21, A is assumed to be the input, B the output, and K corresponds to the gate terminal in a MOS transistor defined as the key in wGS networks. K is assumed to be another input to the switch.

Some Definitions

Definition 5.4 - The switch "S" in Figures 21 and 22 is said to be "ON" when the key "K" is set to logic value "1" in the positive switch ("0" in the negative switch). At this state the input of the switch is connected to the output, thus point B (D) has the same logic value as A (C).

Definition 5.5 - The switch "S" in Figures 21 and 22 is said to be "OFF" when the key "K" is set to logic value "0" in the positive switch ("1" in the negative switch). At this state there is no relation between input and output of the switch and line A (C) is disconnected from line B (D).

Definition 5.6 - The output of a "wired-Gate" device is acceptable only with the following input conditions:

1) The two inputs have the same logic values
2) One input has logic value "Z" (high impedance) and the other a level of logic value "1" or "0" (e.g. "1","1^W","0","0^W")
3) One input is weaker than the other for any logic value.

A "wired-Gate" symbol and its logic function (truth table) for $V6=(0,0^W,1,1^W,Z,U)$ is shown in Figure 23(a) and 23(b), respectively.

In the wGS network structure:

1) Every VDD and GND are assumed to be primitive inputs.
2) The output of every switch or "wired-Gate" device is input(s) to some other switch(es) and/or "wired-Gate" element and/or primitive output.
3) Each input to a "wired-Gate" element, or key and input of a switch, is either a primitive input or output of another "wired-Gate" or switch.
4) For every primitive input, there is at least 1 path to one or more primitive output(s).
5) For every primitive output, there is at least 1 path to one or more primitive input(s).
6) In switches, for testing purposes, the signal path either travels from input to output or from key to output or both.

Testing a Switch

To test a switch for "stuck-open" fault in a wGS network, the following steps should be followed:

1) Activation - Set the key(control) in the positive(negative) switch to logic value "1"("0").
2) Propagation - Propagate signal "s" from the input of the switch to a primitive output of the network. The signal "s" represents either logic value "0" or logic value "1."
3) Implication - Justify other lines in the network by using the backtracking approach from the input and from the key of the switch to the primitive inputs of the network.

When a switch in a network is used to pass both logic values "0" and

"1", the above process must be repeated twice (i.e. signal "s" in Step 2 will be set to "1" in one trail and to "0" in another trail). Otherwise, signal "s" in Step 2 will be set to the logic value that the switch is used for.

To test a switch for "stuck-short", set the key in the positive(negative) switch to logic value "0"("1") and set the input to the logic value opposite of the current logic value of the output, then repeat Steps 2 and 3. In this case, the propagation in Step 2 starts from the output of the switch rather than the input.

The following example illustrate the point. Figure 28 shows a bridge circuit diagram and its equivalent logic gate level. Using path sensitization, the test set needed to examine this circuit is shown in Table V. The wGS implementation of this circuit is shown in Figure 29. Transistors T_1-T_4 (Figure 28) are single direction and the direction of a signal through these transistors is defined by the path "eahf" and "ebhf" from e to f. These transistors are implemented by switches S_1-S_4 in the equivalent wGS network (Figure 29). The transistor T_5, on the other hand, is bidirectional and the direction of signals through this transistor is defined by the paths "eabhf" and ebahf" from e to f. This transistor is implemented and replaced with a positive bidirectional switch S_5 in the wGS network of Figure 29. A test that detects "stuck-open" and "stuck-short" faults in switch S_5, and also detects these faults in transistor T_5.

It is very simple to see that nodes a and b in circuit of Figure 28(b) act like fanout and/or wired-logic depending upon the input condition of A, B, C, D, and E. Node a in Figure 28(b) is replaced with fanout node c and "wired-Gate" device G_1 in Figure 29, and node b is replaced with fanout node d and "wired-Gate" device G_2. The "wired-Gate" G_3 in Figure 29 represents node h in Figure 28(b) and G_4 represents node f. The pull-up transistor in the circuit is implemented by a logic value "1^w" and replaced in the wGS network. Node e is fanout in both figures and stays the same.

To generate a test for detecting "stuck-open" fault in switch S_5, first, the key E belonging to switch S_5 must be set to logic value "1." Then one of the inputs of switch S_5 (e.g. line 1) must be propagated to the output of the network (the primitive output f). In order to do this, input 2 of "wired-Gate" G_2 must be set to logic value "Z" which results in the key D, belonging to switch S_4, having a logic value of "0." The output of G_2 must be propagated through switch S_3. Since S_5 has been used for this signal already, the signal cannot be propagated to line 3 of switch S_5. Additionally, the key C belonging to switch S_3 must be set to logic value "1." Input 1 of G_3 must be set to logic value "Z" in order to propagate the output of S_3 to the output of G_4 and primitive output f; the key A of switch S_1 will need to be set to the logic value "0."

Next, the input 1 of switch S_5 must be backtracked to the input(s) (primitive input(s)) of the network, which in this case is GND. The input 1 of S_5 can be backtracked through switch S_2 to the node e and GND. Thus, the logic value for primitive input B, which is the key of switch S_2, must be set to logic value "1." Therefore, the test that detect "stuck-open" in switch S_5 and corresponds to transistor T_5 is ABCDE=01101. This test also detects "stuck-open" for switches S_2 and S_3.

To generate a test for detecting "stuck-short" fault in switch

TABLE V

TEST SET FOR THE BRIDGE CIRCUIT AT THE LOGIC GATE LEVEL USING PATH SENSITIZATION

Test No	Inputs A	B	C	D	E	Output F
1	0	0	1	1	1	1
2	1	1	0	0	1	1
3	0	1	0	1	0	1
4	1	X	0	1	0	0
5	0	1	0	1	1	0
6	0	1	1	X	0	0
7	0	0	1	1	1	0

TEST VI

TEST SET FOR THE BRIDGE CIRCUIT USING wGS NETWORK FOR "STUCK-OPEN" AND "STUCK-SHORT" FAULTS

Test No	Inputs A	B	C	D	E	Output F
1	0	1	1	0	1	1
2	0	1	1	0	0	1
3	1	0	0	1	1	0
4	1	0	0	1	0	1

S_5, the same procedure should be taken, only this time, the key E belonging to switch S_5 must be set to logic value "0" and propagation starts from the output (e.g. line 2). The test that detects a "stuck-short" in switch S_5 and corresponds to transistor T_5 is ABCDE=01100. This test also detects "stuck-short" for switches S_1 and S_4. To generate tests for all faults in the network, this procedure must be taken for every switch and must be considered for both "stuck-open" and "stuck-short" faults. Table VI shows the test set needed to detect all of the "stuck-short" and "stuck-open" faults in the circuit of Figure 28(b) and Figure 29.

Critical Path Test Generation at the Transistor Level

In this section a procedure will be presented which is called Critical Path Test Generation at the Transistor Level (CPTGTL). This procedure is similar to the critical path test generation technique at the logical gate level. The CPTGTL is not fault oriented in that it does not attempt to derive test for specific faults. The CPTGTL is intended to detect stuck-switches faults along a sensitized path. Therefore, it would seem desirable to generate tests using the longest possible path throughout the circuits to minimize the number of tests. Starting from the output, with the logic value "0" or "1", a path is traced backwards toward the input(s) in a wGS network. At each element, when the output value is specified, both input (in switches the control line is assumed to be an input) values are specified in such a way that the value of one or both inputs is critical (i.e., if the value of the critical input changes, the output of the element will change). In essence, CPTGTL attempts to define single sensitized paths by driving the critical output back to the primary input(s) of the circuits.

When the output of positive (negative) switches is critical-zero (0^c) or critical-one (1^c), the input of the switches has the same critical value, and the control input has a value of critical-one (critical-zero). It should be noted that since the bidirectional positive (negative) switches for a particular signal act exactly like the single direction positive (negative) switches, whenever the word switch(es) is used, it indicates both the single and/or bidirectional switch(es). The critical value for logic High-Impedance (Z) is either High-Zero-Impedance (Z^0) or High-One-Impedance (Z^1) (the difference between the two will be explained later). When the output of a positive (negative) switch is Z^0, the input of the switch has logic value "0" and the control input has critical-zero (critical-one), for the output Z^1; the input has logic value one and the rest is the same.

If the output of the "wired-Gate" devices is critical-one (critical-zero), one of the inputs has critical-one (critical-zero) values and the other has either critical-one (critical-zero), logic-one (logic-zero), weak-zero (weak-one), Z, or Z^0 (Z^1). Depending upon the structure of the circuit, if the output of a "wired-Gate" element is set to logic value critical-one (critical-zero), it is possible that one of the inputs to the same element has value Z^0 (Z^1). If the output of the "wired-Gate" devices is set to Z^0 (Z^1), the inputs must be set to logic value Z^0 (Z^1) too.

Definition 5.7 - A line (node) "L" having critical High-Impedance is said to be High-One-Impedance, if in a faulty circuit "L" has logic value "1".

Definition 5.8 - A line (node) "L" having critical High-Impedance is said to be High-Zero-Impedance if in a faulty circuit "L" has logic value "0".

It should be noted that VDD and GND are assumed to be logic value critical-one (1^c) and critical-zero (0^c), respectively. Figure 30 shows the logic functions (sensitized functions) for switches and "wired-Gate" elements in wGS networks when the output of elements has critical values. The selected values for input A in part(c) of both Figure 30 and Figure 23 are the values that are normally selected during a test generation process. The allowed values can be used if line A is already set to the logic value specified in that column for a specific output value (line C), while the test generation is in progress.

The following example illustrates the point. Figure 31 is the wGS implementation of 2-input NMOS "NOR" gate of Figure 13. Initially set $e=0^c$. To define critical inputs to G_2, d will be set to 0^c. To define critical inputs to G_1 we set $b=0^c$, c=Z or b=Z, $c=0^c$. Considering the first alternative, $A=1^c$ and B=0; switch S_1 is tested for stuck-open. Considering the second alternative, A=0 and $B=1^c$, switch S_2 is tested for stuck-open. In the first alternative, edba and edbA are the sensitized paths. In the second alternative, edca and edcB are the sensitized paths. Line c in the first alternative and line b in the second alternative are the justification part of the test generation. In this case setting c and b to the critical values does not serve any purpose. An additional test can be generated by initially defining the output $e=1^c$. To define critical inputs to G_2, since one input already set to "1^w", that indicates (see Figure 30) d must be Z^0. To define critical inputs to G_1, since $d=Z^0$, therefore,

Input A	Control K	Output B
1^c	1^c	1^c
0^c	1^c	0^c
0	0^c	z^0
1	0^c	z^1

(a) Positive Switches

Input A	Control K	Output B
1^c	0^c	1^c
0^c	0^c	0^c
0	1^c	z^0
1	1^c	z^1

(b) Negative Switches

Inputs			Output
A		B	C
Allowed	Selected		
$z^0,1^c,1$	z	1^c	1^c
------	z^0	1^w	1^c
$z^1,0^c,0$	z	0^c	0^c
------	z^1	0^w	0^c
------	z^0	z^0	z^0
------	z^1	z^1	z^1

(c) "wired-Gate" Elements.

Figure 30. Critical Values of wGS Elements (Sensitized Functions).

$b=c=Z^0$ and G_1 has both inputs critical. The outputs of both switches S_1 and S_2 are Z^0, which indicate that $A=B=0^c$. The sensitized path has been driven back to the inputs and both switch S_1 and S_2 have been tested for the stuck-short fault. Thus, the three tests (A,B)=(1,0),(0,1), and (0,0) are generated.

In Chapter III, it has been shown that in some MOS circuits, especially those with CMOS structure, some of the lines (node) act like memory storage. In this type of circuit, to detect some of the stuck-open faults, a preinitialization test(s) is needed to charge or discharge the memory node(s) before running the actual test.

Example: Figure 32 is the wGS representation of the 2-input CMOS "NOR" gate. Initially set $h=0^c$. To define critical inputs to G_2, d will be set to 0^c and e to Z^1. Following the procedure in the previous example for pull-down, $b=0^c$, $c=Z$, $B=0$ and $A=1^c$. To backtrack $e=Z^1$ to primary inputs, B must be changed to 0^c and f must be set to logic value "1." The generated test is AB=10 which detects S_1 for stuck-open and S_3 for stuck-short faults. Using the same procedure the test AB=01 detects S_2 for stuck-open and S_4 for stuck-short. However, since the previous test AB=10 is to set line h=0, the test AB=01 will not detect stuck-open faults at switch S_2 (refer to Chapter III). Therefore, a preinitialization test is needed to charge node h to logic value "1" before running the test AB=01. Preinitialization must be done through node e, since node d cannot be set to logic value "1". The preinitialization test for setting e and therefore h to logic value 1 is AB=00.

The requirement of preinitialization in the wGS networks can be recognized a priori in certain locations, when the "wired-Gate" devices

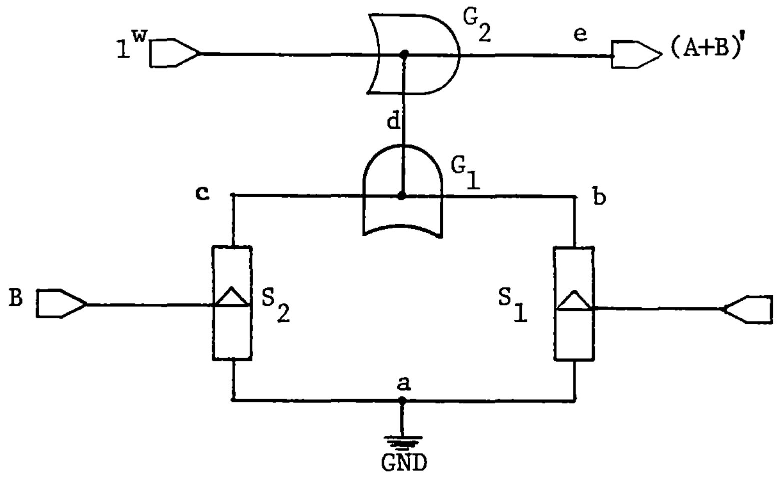


Figure 31. The wGS Representation of the 2-Input NMOS "NOR" Gate

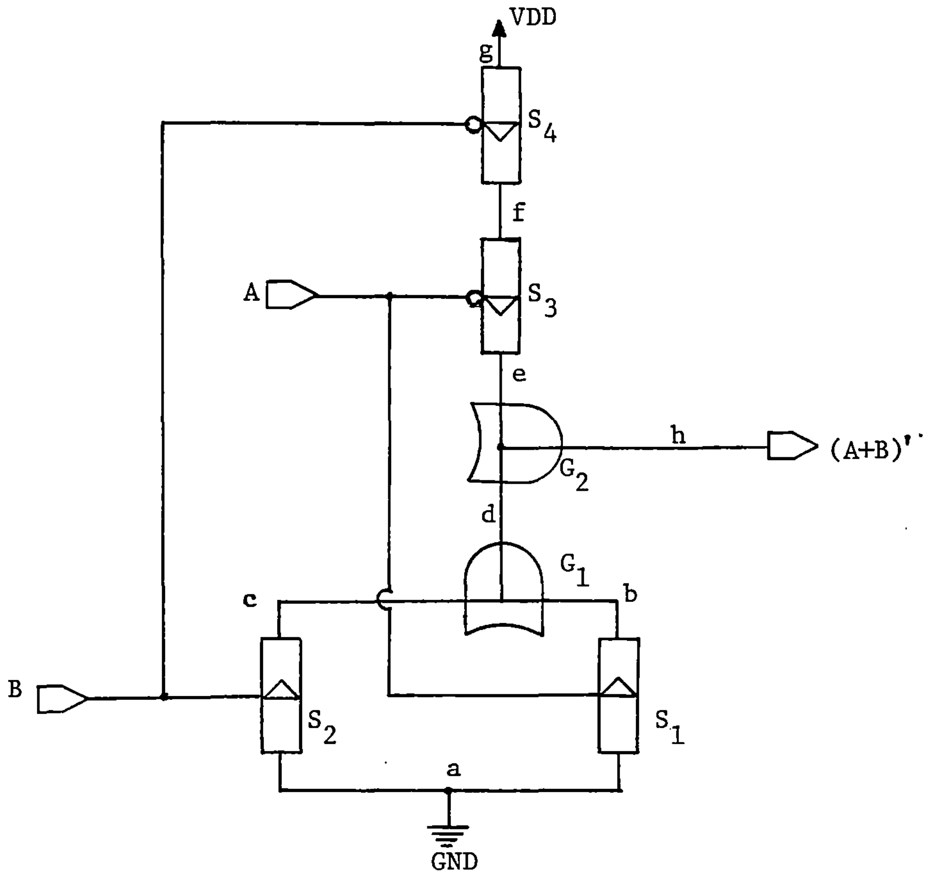


Figure 32. The wGS Representation of the 2-Input CMOS "NOR" Gate

are used to serve specific purposes. In the previous example, the "wired-Gate" G_2 is the device with the specific purpose (complementary wired logic); at G_2 preinitialization is required. Although the "wired-Gate" device introduced in Figure 23(c) shows its behavior with respect to all V5 logic value as inputs, not every "wired-Gate" element in wGS is used to realize every possible function. Table VII shows every possible and specific way that "wired-Gate" elements are used for different allowed inputs in wGS networks; in every case, the input(s) of the element that may need to be preinitialized is specified.

The Critical Path Test Generation algorithm for combinational MOS circuits is thus as follows:

Procedure (Critical Path Test Generation at the Transistor Level)

1) Select an output line. Define it as critical-zero and drive this critical value back towards the primary input using the sensitized function described in Figure 30. Whenever a choice exists, select one alternative and go back for other alternatives later, for additional tests. During the sensitized process conflict might occur; in that case return to the first wG element and try other alternative; continue the process until a choice exist. When all critical lines have been driven back, all non-critical lines are justified using switches and "wired-Gate" functions (primitive function) shown in Figures 21 and 23(c). During the justification process, conflict might occur and more critical lines might be set as a result of a conflicting process. If the conflict cannot be resolved by choosing other alternative, nonexisting of the test and possible redundant circuit is

TABLE VII

POSSIBLE FUNCTIONAL USE OF "WIRED-GATE" DEVICES IN wGS NETWORKS

No.	Input A	Input B	Output C	Preinitialization[1] Description
1	0,Z	0,Z	0,Z	-----
2	0,Z	0,1	0,1	Output C needs to be initialized to 1 through input B when input A is tested for 0^c
3	0,Z	0,1,Z	0,1,Z	
4	1,Z	1,Z	1,Z	-----
5	1,Z	0,1	0,1	Output C needs to be initialized to 0 through input B when input A is tested for 1^c
6	1,Z	0,1,Z	0,1,Z	
7	0,1,Z	0,1,Z	0,1,Z	-----
8	0,Z	1,Z	0,1,Z	Output C needs to be initialized to 1 through input B when input A is tested for 0^c. Output C needs to be initialized to 0 through input A when input B is tested for 1^c
9	0,1	0,1	0,1	-----
10	0,1	0,1,Z	0,1	
11	1^w	0,Z	0,1	-----
12	0^w	1,Z	0,1	-----

Note: Inputs A and B are symmetric.

[1] The input(s) lines that may need to be preinitialized are indicated.

indicated. Every switch along the sensitized path is marked for the fault that the switch has been tested for, and every wired-Gate along the sensitized path is marked for the critical value of the line through the wG element.

2) If a node, during processing Step 1, needs to be preinitialized, justify the node (for preinitialized value) to primary inputs using primitive functions. Generate preinitialization test(s) before continuation of the actual tests in Step 1.

3) Repeat for the output line initialed as a critical-one.

4) If every switch along the paths ending to the primitive output selected in Step 1 is tested for both stuck-short and stuck-open faults and every "wired-Gate" element in these path is tested for all critical values, then remove this particular primitive output from the wGS network.

5) Repeat steps 1, 2, 3 and 4 for the rest of the output lines.

6) Repeat steps 1, 2, 3, 4 and 5 until all of the output lines are removed from the network.

Although this method will detect all of the stuck-switches faults along the sensitized paths, the algorithm is designed to detect only the single stuck-switches faults. However, along multiple sensitized paths, some multiple faults might be detected, though they are not determined.

Example: Figure 33 is the wGS representation of NMOS Selector Logic Circuit shown in Figure 18. The table of Figure 34 shows the computation for the critical path test derivation procedure for this circuit. Output of each device is defined with name of that element. Start with a critical-zero (0^c) on line G_5. We now have a choice to

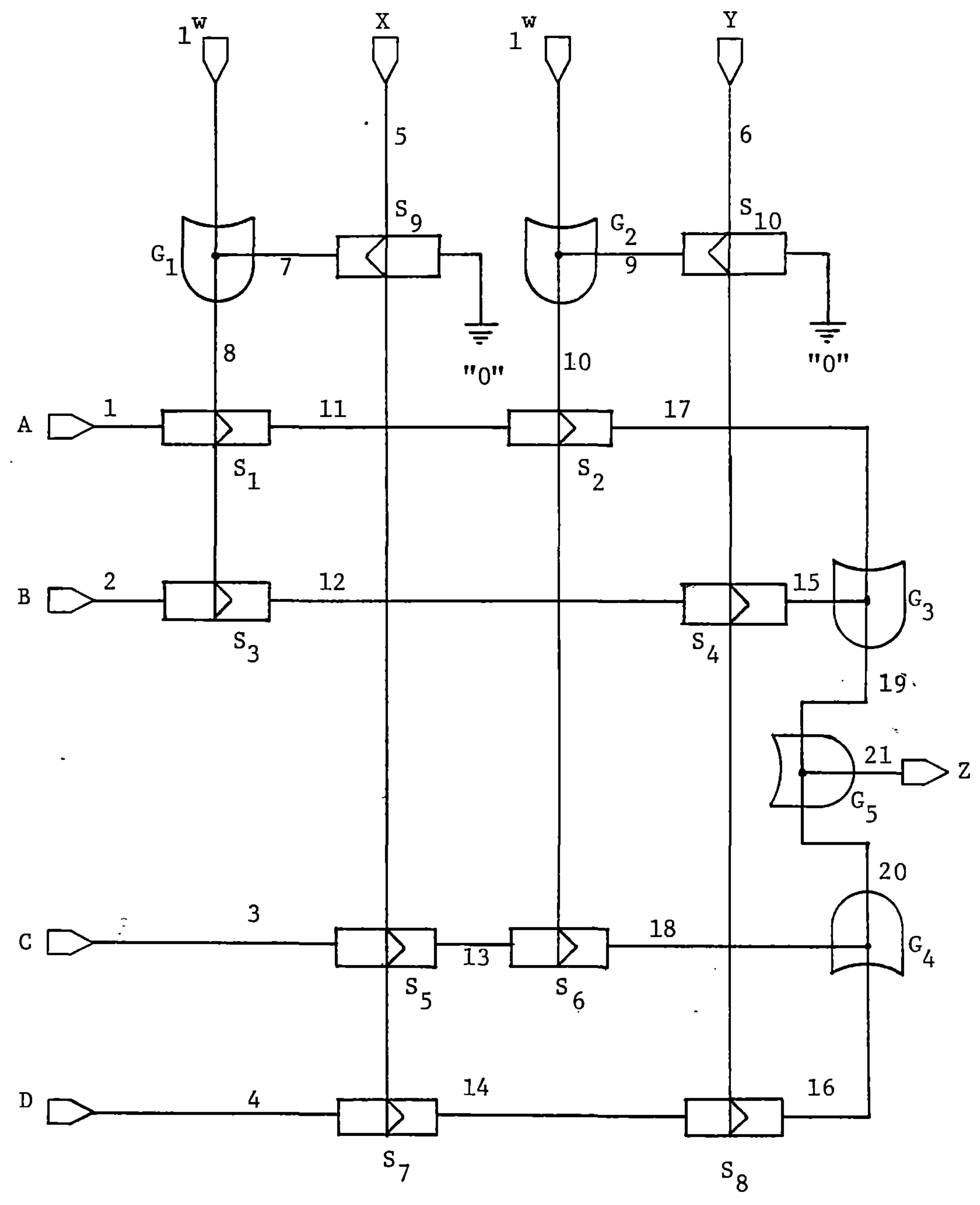


Figure 33. The wGS Representation of NMOS Slector Logic Circuit of Figure 18

Row	A	B	C	D	X	Y	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	G_1	G_2	G_3	G_4	G_5
1																					0^c
2																				0^c	0^c
3														0^c						0^c	0^c
4						1^c							0^c	0^c						0^c	0^c
5				0^c	1^c	1^c							0^c	0^c						0^c	0^c
6	X	1	1	0^c	1^c	1^c	z	z	z^1	z^1	1	z^1	0^c	0^c	0^c	0^c	0^c	0^c	z	0^c	0^c
7				1^c	1^c	1^c							1^c	1^c						1^c	1^c
8	X	0	0	1^c	1^c	1^c	z	z	z^0	z^0	0	z^0	1^c	1^c	0^c	0^c	0^c	0^c	z	1^c	1^c
9			0^c		1^c	0^c					0^c	0^c				z^0		1^c		0^c	0^c
10	1	X	0^c	1	1^c	0^c	z^1	z^1	z	z	0^c	0^c	1	z^1	0^c	z^0	0^c	1^c	z	0^c	0^c
11			1^c		1^c	0^c					1^c	1^c				z^0		1^c		1^c	1^c
12	0	X	1^c	0	1^c	0^c	z^0	z^0	z	z	1^c	1^c	0	z^0	0^c	z^0	0^c	1^c	z	1^c	1^c
13		0^c			0^c	1^c			0^c	0^c					z^0		1^c		0^c		0^c
14	1	0^c	X	1	0^c	1^c	1	z^1	0^c	0^c	z	z	z^1	z^1	z^0	0^c	1^c	0^c	1^c	z	0^c
15		1^c			0^c	1^c			1^c	1^c					z^0		1^c		1^c		1^c
16	0	1^c	X	0	0^c	1^c	0	z^0	1^c	1^c	z	z	z^0	z^0	z^0	0^c	1^c	0^c	1^c	z	1^c
17	0^c				0^c	0^c	0^c	0^c							z^0	z^0	1^c	1^c	0^c		0^c
18	0^c	1	1	X	0^c	0^c	0^c	0^c	1	z^1	z^1	z^1	z	z	z^0	z^0	1^c	1^c	0^c	z	0^c
19	1^c				0^c	0^c	1^c	1^c							z^0	z^0	1^c	1^c	1^c		1^c
20	1^c	0	0	X	0^c	0^c	1^c	1^c	0	z^0	z^0	z^0	z	z	z^0	z^0	1^c	1^c	1^c	z	1^c

Figure 34. The Computation for the Critical Path Test Derivation Procedure for Circuit in Figure 33.

set either G_3 or G_4 to 0^c, G_4 is set to 0^c (second row of table in Figure 34). Line G_4 having 0^c implies that either S_6 or S_8 is 0^c; for this test S_8 is chosen to have 0^c (row 3 of the table). Line $S_8 = 0^c$ implies that $Y = 1^c$ and $S_7 = 0^c$ (fourth row of the table). Line $S_7 = 0^c$ implies that $X = 1^c$ and $D = 1^c$. At this point all lines for the critical path are identified (row 5 of the table). Next, the other lines in the circuit must be justified.

Lines $X=Y=1^c$ implies that lines $S_9=S_{10}=G_1=G_2=0^c$. Backtracking from G_4, one of the inputs of wired-Gate G_4 is set to 0^c we first set the other input line S_6 to Z. Since the control line to the switch S_6 is 0^c and there is a path from input of S_6 to primary input C (S_5 is ON), therefore, we can change line $S_6=Z^1$ which implies that $S_5=C=1$. Starting from G_5, $G_5=G_4=0^c$ indicates that $G_3=S_4=S_3=Z$ $(Y=1^c)$. However, since in switch S_3 the control line G_1 is 0^c and B is a primitive input, therefore B=1 and $S_3=S_4=Z^1$ (they changed from Z to Z^1). The last justifying is started from wired-Gate G_3 to switches S_2 and then S_1. Line $G_3=Z$ indicates that $S_2=Z$. Although $G_2=0^c$, since there is no path from the input of switch S_2 to the primary input or a line with value 1, S_2 cannot be set to Z^1. Therefore the path with A=X (don't care) and $S_1=Z$ stops here. The implication part of this test is shown in row 6 of Figure 34. The complete derivation of critical tests and implication of other tests for this circuit are shown in rows (7,9,11,13,15,17,19) and (8,10,12,14,16,18,20), respectively.

Summary

An extension of the existing method of generating tests at the logical gate level is introduced in this chapter to generate tests for

MOS digital circuits at the transistor level. The CPTGTL (Critical Path Test Generation at the Transistor Level) uses sensitized paths throughout the wGS network to detect faults. The CPTGTL uses stuck switches faults to generate tests. A faulty switch in the wGS networks is either permanently stuck-open or permanently stuck-short. To identify a faulty switch along a sensitized path in the wGS implementation of a MOS circuits, additional logic values, High-Zero-Impedance and High-One-Impedance (critical high-impedance), have been defined in this chapter.

The procedure of the CPTGTL is similar to critical path procedure at the logical gate level:

a) Both methods use sensitized path to detect faults.

b) The Critical Path Test Generation (CPTG) at the logical gate level uses sensitized-cubes and primitive-cubes to process test generation [10]. The CPTGTL, on the other hand, uses the sensitized-functions and primitive-functions described in this chapter and Chapter IV.

c) The CPTGTL uses stuck-switches faults and CPTG uses stuck-at faults to detect failures in a circuit.

However, the CPTGTL generates tests to charge or discharge nodes which act like a memory storage in certain parts of some MOS circuits.

Appendix A contains the listing of a PL/1 computer program which is written on an IBM 3081K to implement the CPTGTL algorithm. This program is used to generate tests for the last example in this chapter (Selector Logic Circuit shown in Figure 33) and few more examples shown in Appendix B. The input networks for these examples and the output generated tests are also presented in Appendix B.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

The structure and elements of digital circuits at the logical level and layout level of MOS technology are different. Gates, which are the elements at the logical level of digital circuits, and switches (elements at the layout level of MOS type digital circuits) do not have the same physical structure. The conventional gate level structures are designed to represent logic function. However, a gate level diagram may not accurately represent the topology of the transistor level implementation of particular MOS digital circuits. The path sensitization method at the logical gate level uses stuck-at fault models to detect faults, but in some MOS circuits stuck-open faults also need to be detected. Since the logical gate level structures do not accurately describe the topology of MOS type design, detection of specific short circuit faults is difficult at the gate level. As a result, the available test generation methods at the logical gate level are not completely adequate for testing MOS digital integrated circuits.

It has been shown in Chapter II that the CSA switching theory implements the structural properties of MOS type design. However, the bidirectional lines and switches in CSA networks make it quite difficult to be used for test generation purposes. The wGS networks, a modified version of CSA switching theory with imposed directionality, are designed to implement MOS structures for test generation purposes. Path sensitization can be used to generate tests for wGS networks. The stuck-switches fault models have been defined in Chapter V. This model is used for detection of faults in the wGS networks. The faults are permanently stuck-open or stuck-short. Additional logic values besides those used at the logical gate level are defined for use in testing the circuits at the transistor level. The method for testing the wGS network has been explained.

The Critical Path Test Generation at the Transistor Level (CPTGTL) was introduced in Chapter V. The CPTGTL use sensitized paths to generate tests. Besides faults found to correspond to the stuck-at fault at the logical gate level, the CPTGTL generates tests to detect stuck-open faults in MOS circuits. The CPTGTL does not use the sensitized-cubes or primitive-cubes; instead, it uses sensitized and primitive functions defined in this research. Along with the CPTGTL procedure, a PL/1 computer program is written and presented in Appendix A to implement the CPTGTL algorithm. Appendix B contains input network descriptions and the result of using the PL/1 test generation program on several examples.

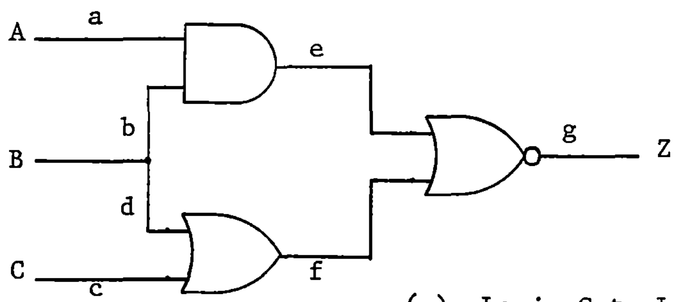
Recommendations for Further Research

Several areas of further research have been established for testing MOS digital systems at the transistor level. The effort required to construct the network description by adding the "wired-Gate" elements and the restructuring of three-terminal bidirectional MOS transistors to five-terminal switches might be considered a disadvantage by the users. In order to provide a wGS network from a circuit level description of a particular MOS digital system, one must replace all of the wired-logic nodes with one or more "wired-Gate" devices, restructure the bidirectional switches, and identify the direction of every single line in a circuit.

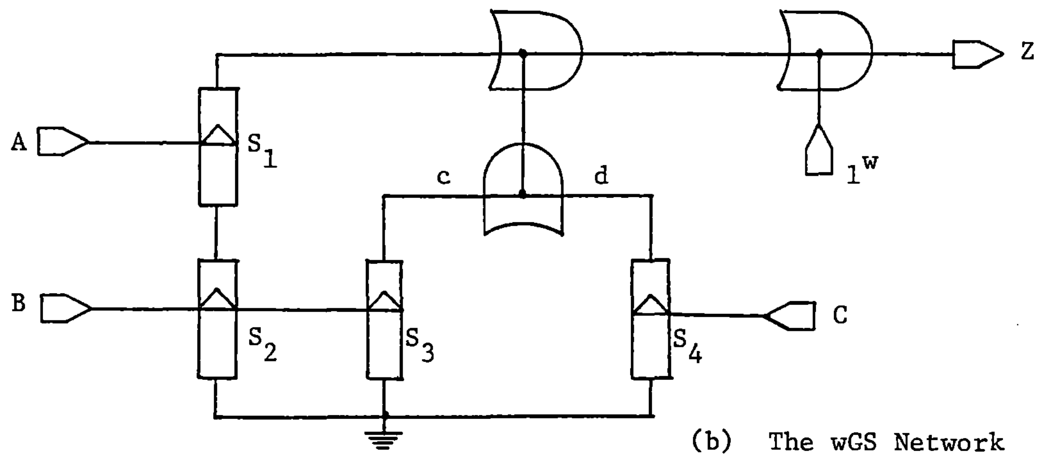
This process can be achieved more easily with the help of a computer algorithm to realize the design methods explained in Chapter IV and the information shown in Table VII. To approach this task, one might need to provide more additional information, along with the transistor level networks, to make the complicated parts of a circuit more recognizable. As a result, developing a method that preprocesses a transistor level implementation of an MOS circuit to a corresponding wGS network seems to be a useful area of future research.

Although the CPTGTL detects some of the short circuit faults in the wGS networks, this algorithm is not designed to generate tests for specific short-circuit faults. It is quite difficult and practically impossible to detect every possible short-circuit fault. However, it does not seem to be as impractical to develop a separate procedure to generate tests for specific short-circuit faults using wGS implementation of circuit layout. Therefore, a test generation method for detecting specific short-circuit faults, using the wGS representation of a particular MOS circuit, might be a good area for further research.

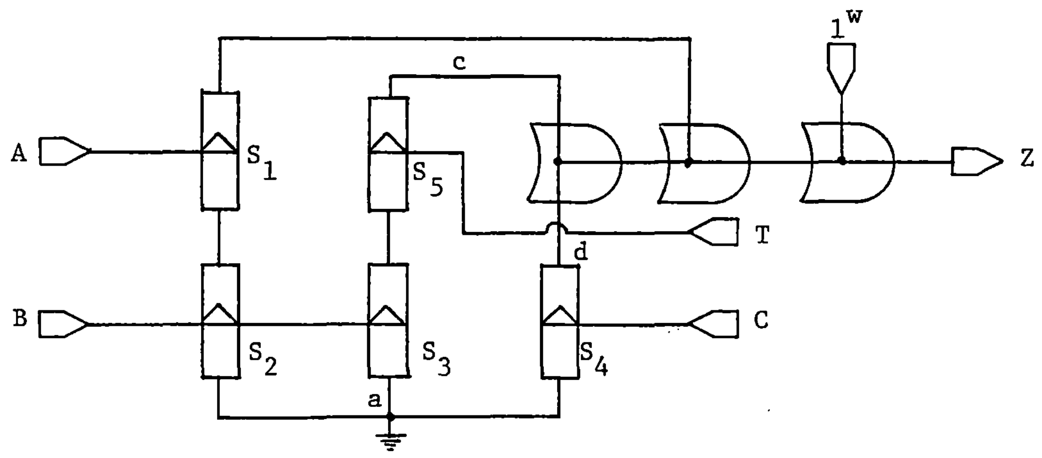
Like critical path test generation at the logical gate level, the CPTGTL may not generate tests for some of the faults in redundant circuits. The circuit in Figure 35 illustrates an extremely simple redundant circuit. Part (a) represents the logical gate level of the circuit. Using a stuck-at model, line A cannot be tested for either



(a) Logic Gate Level



(b) The wGS Network



(c) Design Method

Figure 35. A Redundant Circuit

s-a-1 or s-a-0. Line B cannot be tested for s-a-0. The wGS implementation of this circuit is shown in part (b) of this figure. In this network, switch S_1 cannot be tested for either stuck-open or stuck-short and switch S_2 cannot be tested for stuck-open. To detect S_1 for stuck-short (closed), the switches S_1, S_3, and S_4 must be "OFF" and S_2 must be "ON." Since both S_2 and S_3 are controlled by line B, then whenever S_2 is "ON" ("OFF") S_3 is also "ON" ("OFF") and vice versa. To detect S_1 for stuck-open, the switches S_1 and S_2 must be "ON" and S_3 and S_4 must be "OFF." In both cases the contradiction of state in the switches S_2 and S_3 results untestability of S_1 for stuck-open and stuck-short. To detect S_2 for stuck-open, the switches S_3 and S_4 must be "OFF" and S_1 and S_2 must be "ON." Thus the contradiction of setting the switches S_2 and S_3 to the proper state results in S_2 being not testable for stuck-open.

The problem of testing the redundant circuits can be solved with proper and appropriate design method. The use of dummy switches (transistors) in certain parts of the redundant lines might solve this problem. These dummy switches are functional only at the time of testing the circuits; they stay permanently "ON" or permanently "OFF" after the circuits have been tested. Consider part C of Figure 35. This circuit is a remodeled version of the circuit in part (b) for testability. If the switch S_5 (the dummy switch used for testing purposes) stays permanently "ON", this circuit functions exactly like the original network shown in part (b). During testing the circuit, S_5 plays the auxiliary switch for S_3 to disconnect the path ac by setting the line T (input-test line) to logic value "0." Normally, line

T has logic value "1." This line will be changed to "0" when S_2 is "ON" and S_3 is "OFF." For example, when we test the switch S_1 for stuck-short, S_2 must be "ON" and S_3 must be "OFF" to open the path from a to c. However, whenever S_2 is "ON", S_3 is"ON" too. Therefore, to open the path ac the auxiliary switch S_5 will be used and line T will be set to logic value "0".

Thus, the testability problem in the redundant circuits can be solved by using the proper design method. Detecting the redundant lines and using the proper structure to remodel the circuits for testing purposes seems to be a good area for further research in testing MOS digital circuits at the transistor level.

REFERENCES

[1] El-Zig, Y. M., Y. H. Su, "Fault diagnosis of MOS combinational network," IEEE Trans. on Comp., Vol. C-31, No. 2, February 1982.

[2] Siewiorek, D. P., L. K. Lai, "Testing of digital system," Proc., of IEEE, Vol. 69, No. 10, October 1981.

[3] Eichelberger, E. B., and T. W. Williams, "A logic structure for LSI testability," in proc. 14th DEsign Automation Conf., New Orleans, pp. 462-468, June 1977.

[4] Chappell, S. G., P. R. Menon, J. F. Pellegrin, and A. M. Schowe, "Functional simulation in the LAMP system," J. Des. Automat. Fault-Tolerant Comput., Vol. 1, pp. 203-216, May 1977.

[5] Duley, J. R., "DDL - A digital system design language," Ph.D. dissertation, Univ. of Wisconsin, Madison, 1967.

[6] Wadsack, R. L., "Fault modeling and logic simulation of CMOS integrated circuits," BSTJ, Vol. 57, pp. 1449-1474, May/June 1978.

[7] El-Zig, Y. M., "Testing of MOS combinational network: a procedure for efficient fault simulation and test generation," Proc 16th Design Automation Conf., San Diego, pp. 162-170, June 1979.

[8] Seller, Jr., F. F., M. Y. Hsiao, and L. W. Bearnson, "Analyzing errors with the Boolean difference," IEEE Trans. Comput., Vol. C-17, pp. 676-683, July 1968

[9] Susskind, A. K., "Additional applications of the Boolean difference to fault detection and diagnosis," in 1972 Int. Symp. Fault Tolerant Computing, 1972, pp. 58-61

[10] Breuer, M. A., A. D. Friedman, _Diagnosis and Reliable Design of Digital Systems_. Potomac, MD: CSPI, 1976.

[11] Roth, J. P., "Diagnosis of automata failures: A calculus and a method," IBM J. Res. Develop., Vol. 10, pp. 278-291, July 1966.

[12] Roth, J. P., W. G. Bouricus, and P. R. Schneider, "Programmed algorithms to computer tests to detect and distinguish between failures in logic circuits," IEEE Trans. Comput., Vol. C-16, pp. 567-580, Oct., 1967.

[13] Shannon, C. E., "A symbolic analysis of relay and switching networks," Trans. IEEE, Vol. 57, pp. 713-723, 1938.

[14] Hayes, J. P., "A unified switching theory with applications to VLSI design," Proceedings of the IEEE, Vol. 70, No. 10, pp. 1140-1151, October 1982.

[15] Harrison, M. A., Introduction to Switching and Automata Theory. New York:McGraw Hill, 1965.

[16] Murogo, S., Logic Design and Switching Theory. New York:Wiley-Interscience, 1979.

[17] Lofgren, L., "A theory of uniform switching nets," University of Illinois, Elec. Eng. Res. Lab. Tech. Rep., May 6, 1962.

[18] Mead, C., and L. Conway, Introduction to VLSI System. Reading, MA:Addison-Wesley, 1980.

[19] ISD Inc., LOGIS User's Manual. Santa Clara, CA, circa 1978.

[20] Hayes, J. P., "A logic design theory for VLSI," presented at the 2nd Caltech Conf. on VLSI, Pasadena, CA, Jan. 1981, (Proceedings to appear).

[21] Levendel, Y. L., P. R. Menon, and C. E. Miller, "Accurate logic simulation models for TTl totempole and MOS gates and tristate devices," Bell Yyst. Tech. J., Vol. 60, pp. 1271-1287, Sept. 1981.

[22] Hayes, J. P., "A fault simulation methodology for VLSI," in Proc. 19th Design Automation Conf. (Las Vegas, NV), pp. 393-399, June 1982.

[23] McCluskey, E. J., Fellow, "C/D Tester for CMOS Stuck-Open-Faults," IEEE Trans. Comput., Vol. c-30, pp. 873, Nov. 1981.

[24] Bhattacharya, E. B., and B. Gupta. "Logical Modeling of Physical Failures and their Inherent Syndrome Testability in MOS LSI/VLSI Networks," IEEE International Test Conference, Philadelphia, PA., 1984.

[25] Singer, D. M. "Testability Analysis of MOS VLSI Circuits," AT&T Bell Laboratories, Murray Hill, New Jersey 07974. IEEE International Test Conference, Philadelphia, PA. 1984.

[26] Banerjee, P., and J. A. Abraham. "Generating Test for Physical Failures in MOS Logic Circuits." Coordinated Science Laboratory, University of Illinois, Urbana, IL-61801. IEEE International Test Conference, Philadelphia, PA. 1983.

[27] Baschiera, D., and B. Courtois. "Testing CMOS: A Challenge," VLSI Design, pp. 58-62, Oct. 1984.

APPENDIXES

APPENDIX A

LISTING OF TEST GENERATION COMPUTER PROGRAMS

```
/** TGCOMENT **

        ****************************************************************
        *                                                              *
        * PROGRAMMER: M. T. MOSTAFAVI          DATE: 12/04/1985        *
        *                                                              *
        * TITLE: CRITICAL PATH TEST GENERATION FOR "wGS" NETWORKS      *
        *                                                              *
        * REFERENCE:                                                   *
        *     Data Structures and Algorithms by Alfred V. Aho, John    *
        * E. Hopcroft, and Jeffrey D. Ullman.  The IBM PL/1 Language   *
        * Reference Manual.                                            *
        *                                                              *
        *     Program created at  OKLAHOMA  STATE  UNIVERSITY on the   *
        * IBM 3081K using PL/1 Language and compiled using  PL/1       *
        * Optimizing Compiler.                                         *
        *                                                              *
        ****************************************************************

       PURPOSE AND DESCRIPTION OF PROGRAM:

             The purpose of this program is to generate a set of tests
            to detect "stuck-open" and "stuck-short" faults in a
            wGS representation of a MOS circuit.  This program uses
            the Critical Path Test Generation at the Transistor Level
            algorithm to generate tests.

             The program requires a one-character letter as input.
            A "Y" as the input parameter to the main program  represents
            the "trace" option.  Any other character is defined as "no
            trace."  The trace file is defined as "FTRACE."  The minimun
            record length for the trace file is 80.  The DISP for the
            file "FTRACE" defined in the JCL must be "OLD."  The record
            format for the file "FTRACE" in the JCL must be "VBA."

             A tree structure is used to store a wGS network into the
            program.  Every primitive output is assumed to be a root
            to the structure.  Every element (wierd-Gate or Switche) is
            a node.  The node number is defined as the output of the
            element.  A node in the network structure is called "ELEMENT"
            which is described in the declaration part of the program.
            The "CNTLINE" and "INLINE" are the two leaves to each
            element.  The "CNTLINE" corresponds to the control line in a
            switch or one input to the wired-Gate.  The "INLINE"
            corresponds to the input line in a switch or other input to
            the wired-Gate.

             For accuracy, every module of the program has been tested
            individually.  Depending upon the complexity of the modules,
            numerous examples are used to test each module before they
            have been put together for the final run.
                              ---------------------------

       INPUT DESCRIPTION:

             The input file for the program is defined as "IN".  The
            maximum record legnth  for the input file is 1920.  The DISP
            for the file "IN" must be defined as "OLD" or "SHR" in the
            JCL.  The file "IN" must contain the circuit description
            of a wGS network.  The format for a wGS input network is as
            follows:

            - The first line of the input network must be
                   MAX X
```

where X is the maximum number of elements used in the network.

- Primitive Outputs (PO) of the network are defined as
  ```
  PO X 01 02 . . . . . On
  ```
  where X is the number of primitive outputs defined in this line and 01-On are the primitive outputs line number.

- Primitive Inputs (PI) of the network are defined as
  ```
  PI I N
  ```
  where I is the line number for the primitive input and N is the input number.

- Elements in the network are defined as
  ```
  TYPE N I C
  ```
  where TYPE is the type of element used in the network, N is the element number (or output line number), I is the input line number, and C is the control line number for switches and the other input for wG elements.

Types of elements:

PS - A Positive Switch

NS - A Negative Switch

IP - A Positive Switch with the input line connected to GND or VDD. If the input line is connected to GND, then input line number is 0, otherwise is 1.

IN - A Negative Switch with the input line connected to GND or VDD. If the input line is connected to GND, then the input line number is 0, otherwise it is 1.

BP - A Bipositive Switch. To define a Bipositive Switch, two PS must be used. If the first PS is element-no (N) then the second PS must be element-no (N+1). The control line of the second element must be set to value 0. For example, a Bipositive Switch "S" with input 12 and 24, outputs 15 and 16, and control line 25 is defined as:
```
BP 15 12 25
BP 16 24 0
```

BN - Same as BP, but for a negative switch.

WG - A wG element with inputs realizing all possible values 0, 1, Z.

WU - A wG element with one input realizing values (0,z) and the other input realizing the value weak-one (i.e. line used as pull-up). In this element, only the input for two lines (input and output) is needed. For example, WU-no 15 with one input connected to line 13 and the other used as pull-up is defined as
```
WU 15 13
```

WD - A wG element with one input realizing values (1,z) and the other input realizing the value weak-zero (i.e. line used as pull-down). In this element, only input for two lines (input and output) is needed. For example, WU-no 15 with one input connected to line 13 and the other used as pull-down is defined as
```
WD 15 13
```

WC - A wG element with the first input realizing values (0,Z) and the second input realizing values (1,Z). This element replaces the Complimentary Wired-Logic in MOS circuits. It should be noted that the line realizing logic values (0,Z) must be used as the first input. For example, WC-no 15 with input line 13 realizing logic values (0,Z) and input line 12 realizing logic values (1,Z) is defined as

```
WC 15 13 12
```

WL - A wG element with the first input realizing values (0,1,Z) and the second input realizing the values (0, Z), (i.e. the second line must be entered as the last information for this element in the input record). for example, WL 15 with the first input connected to line 13 and the second input connected to line 12 is defined as

```
WL 15 13 12
```

WH - A wG element with the first input realizing values (0,1,Z) and the second input realizing the values (1, Z), (i.e. the second line must be entered as the last information for this element in the input record). for example, WL 15 with the first input connected to line 13 and the second input connected to line 12 is defined as

```
WL 15 13 12
```

- Line started with any other character or word will be assumed to be a comment.

OUTPUT DESCRIPTION:

The output file for the program is defined as "OUT". The minimum record length for the output file is 125. The DISP for the file "OUT" defined in the JCL must be "MOD." The record format for the file "out" in the JCL must be "VBA." The generated tests for the wGS network as described in the file "IN" will be stored in the file "OUT". A record in the file "OUT" is described as follows:

- The first field of the output record is the test number. The value 0 in this field indicates that the test is for setting a node to the logic value "0" or logic value "1".

- The second field of the output record is a number belonging to the test that must be followed after this test.

- The third field of the output record is the output line number of the network for this test.

- The fourth field of the output record is the logic value for the output line in the third field.

- The fifth field, to the end of the record, are logic values (0,1,X) for inputs (1,2,.....,n), in that order.

PROGRAM INTERCONNECTION:

The program connectivity will be described as father and sons. The calling program is defined as father and the

```
          called program is defined as the son. The father and sons
          relationship are as follows:

               FATHER        |                  SONS
        ====================|=========================================
               MAIN          |     TRACE, PLIRETC (system program),
                             |  INPUT_DATA, OUTPUT_DATA, PROCESS_NETWORK
        --------------------|-----------------------------------------
               TRACE         |                  (non)
        --------------------|-----------------------------------------
             INPUT_DATA      |                  TRACE
        --------------------|-----------------------------------------
            OUTPUT_DATA      |                  TRACE
        --------------------|-----------------------------------------
                             |    GENERATE_TEST, FIND_CRITICAL_PATH,
          PROCESS_NETWORK    |     TRACE, FIND_PATH_FOR_INIT_NODE,
                             |           JUSTIFY_OTHER_LINE
        --------------------|-----------------------------------------
           GENERATE_TEST     |                  TRACE
        --------------------|-----------------------------------------
         FIND_CRITICAL_TATH  |                  TRACE
        --------------------|-----------------------------------------
           FIND_PATH_FOR_    |                  TRACE,
                  INIT_NODE  |    GENERATE_TEST, JUSTIFY_OTHER_LINE
        --------------------|-----------------------------------------
                             |  TRACE, CONFLECT_GO, CONFLECT_PROCESS,
         JUSTIFY_OTHER_LINE  |   CONFLECT_RETURN, CONFLECT_IS_RETURN
                             | CONFLECT_WC_PROCESS, CONFLECT_WH_PROCESS
                             | CONFLECT_WL_PROCESS, CONFLECT_WP_RETURN
        --------------------|-----------------------------------------
            CONFLECT_GO      |                  TRACE
        --------------------|-----------------------------------------
          CONFLECT_PROCESS   |                  TRACE
        --------------------|-----------------------------------------
          CONFLECT_RETURN    |                  TRACE
        --------------------|-----------------------------------------
         CONFLECT_IS_RETURN  |                  TRACE
        --------------------|-----------------------------------------
        CONFLECT_WC_PROCESS  |                  TRACE
        --------------------|-----------------------------------------
        CONFLECT_WH_PROCESS  |                  TRACE
        --------------------|-----------------------------------------
        CONFLECT_WL_PROCESS  |                  TRACE
        --------------------|-----------------------------------------
         CONFLECT_WP_RETURN  |                  TRACE
        ==============================================================

** END OF TGCOMENT **/


/**TGMAIN**/
TGMAIN: PROC(INFLAG) OPTIONS(MAIN);

      /***************************************************************
       *                                                             *
       *  The  main  program  reads  the  maximum  elements  in the *
       * network from the wGS network data file, defines storage for *
       * the program,  reads the network data file into the program  *
       * by  calling  the  INPUT_DATA  subprogram,  then starts     *
       * processing  the  test  generation  by  calling  the        *
       * PROCESS_NETWORK  subprogram.  The program  prints a message *
       * if the  input  network is  invalid or if an invalid path is *
       * detected during the test generation process.  If the value  *
       * of the  input  parameter  "INFLAG"  is  "Y"  then the  main *
```

```
        * program sets the trace flag "ON" and calls the OUTPUT_DATA  *
        * routine for printing the input network into the trace  file *
        * (FTRACE).                                                    *
        *                                                              *
        ****************************************************************/


%INCLUDE TGCOMENT;

%INCLUDE TGDCL;

   DCL  INFLAG              CHAR(*) VAR;

   IF SUBSTR(INFLAG,1,1) = 'Y' THEN TRACE_FLAG = YES;
   ELSE TRACE_FLAG = NO;

IF TRACE_FLAG THEN DO;
   OPEN FILE(FTRACE) OUTPUT;
   CALL TRACE('TGMAIN');
END;

   OPEN FILE(IN) INPUT;

   ON ENDFILE(IN) EOF = YES;

   READ FILE(IN) INTO(TEXT);
   COMMAND = SUBSTR(TEXT,1,2);
   TEXT = SUBSTR(TEXT,4) || ' ';

   DO WHILE(COMMAND¬='MA' & COMMAND¬='ma' &
            COMMAND¬='Ma' & COMMAND¬='mA' &
            COMMAND¬='EO' & COMMAND¬='eo' &
            COMMAND¬='Eo' & COMMAND¬='eO' & ¬EOF);
      READ FILE(IN) INTO(TEXT);
      COMMAND = SUBSTR(TEXT,1,2);
      TEXT = SUBSTR(TEXT,4) || ' ';
   END;

   IF EOF | COMMAND='EO' | COMMAND='eo' | COMMAND='Eo' |
                                          COMMAND='eO' THEN DO;
      PUT FILE(SYSPRINT) SKIP EDIT('** EMPTY OR INVALID INPUT FILE.')
                                                                (A);
      CLOSE FILE(IN);
      CALL PLIRETC(1);
      STOP;
   END;

   GET STRING(TEXT) LIST(MAX_ELEMENT);

   ALLOCATE ELEMENT;

   MAX_STACK = MAX_ELEMENT;
   MAX_QUEUE = MAX_ELEMENT;

   ALLOCATE PATH_STACK;
   ALLOCATE PATH_QUEUE;

   ALLOC PRIMITIVE_INPUTS;
   IP = INPUTS_HEAD;
   INPUTS_HEAD->NEXT_IP = NULL;

   ALLOC PRIMITIVE_OUTPUTS;
   OP = OUTPUTS_HEAD;
   OUTPUTS_HEAD->NEXT_OP = NULL;
   CALL INPUT_DATA;
   CLOSE FILE(IN);

   IF TRACE_FLAG THEN CALL OUTPUT_DATA;

   PUT FILE(SYSPRINT) SKIP EDIT('Start processing test generation.')
                                                               (a);
   PUT FILE(SYSPRINT) SKIP(2);
   IF PROCESS_NETWORK THEN DO;
      PUT FILE(SYSPRINT) SKIP EDIT('** INVALID PATH HAS BEEN ',
       'FOUND IN THE NETWORK.')(A,A);
   END;

   PUT FILE(SYSPRINT) SKIP EDIT('Stop processing test generation.')
                                                               (a);
   IF TRACE_FLAG THEN CLOSE FILE(FTRACE);

%INCLUDE TGCONFG;

%INCLUDE TGCONFP;

%INCLUDE TGCONFR;

%INCLUDE TGFCPATH;

%INCLUDE TGFPFIN;

%INCLUDE TGGTEST;

%INCLUDE TGINPUT;

%INCLUDE TGISCFR;

%INCLUDE TGJOPATH;

%INCLUDE TGNETWRK;

%INCLUDE TGOUTPUT;

%INCLUDE TGTRACE;

%INCLUDE TGWCCFP;

%INCLUDE TGWHCFP;

%INCLUDE TGWLCFP;

%INCLUDE TGWPCFR;

END TGMAIN;

/**TGCONFG**/
CONFLECT_GO: PROC(VALUE,VALUE_OUT);

      /***************************************************************
       *                                                             *
       *   This routine calls the next element with the logic value *
       * defined as "VALUE" and changes the output logic value of    *
       * the current element to the logic value defined as           *
       * "VALUE_OUT,"  i.e. as a result of conflect process.         *
       *                                                             *
       ***************************************************************/
   DCL VALUE                 BIT(*),
       VALUE_OUT             BIT(*);

   IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE THEN DO;
      QUEUE_TOP = QUEUE_TOP + 1;
      QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
   END;

   PREDIR(STACK_TOP - 1) = AHEAD;
   OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE_OUT;
   OUT_VALUE(NEXT_ELMT,QSELECT) = VALUE;
   STACK_TOP = STACK_TOP + 1;
   STACK(STACK_TOP) = NEXT_ELMT;
   PREDIR(STACK_TOP) = AHEAD;

IF TRACE_FLAG THEN CALL TRACE('TGCONFG');

END CONFLECT_GO;

/**TGCONFP**/
CONFLECT_PROCESS: PROC(VALUE);

      /************************************************************
       *                                                          *
       *   This routine sets the test-flag for the current element   *
       * and changes the output of the element to the logic value   *
       * defined as "VALUE" (i.e. if the logic value of the current *
       * element is defferent from "VALUE") while returning to the  *
       * previous element.                                         *
       *                                                          *
       ************************************************************/


   DCL VALUE                 BIT(*);

   STACK_TOP = STACK_TOP - 1;
   TEST_FLAG(ELEMENT_NO) = TEST_FLAG(CNTLINE(ELEMENT_NO))
                         & TEST_FLAG(INLINE(ELEMENT_NO));

   IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= VALUE
   THEN DO;
    PREDIR(STACK_TOP) = AHEAD;
    OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;
   END;

IF TRACE_FLAG THEN CALL TRACE('TGCONFP');

END CONFLECT_PROCESS;

/**TGCONFR**/
CONFLECT_RETURN: PROC(FLAG,VALUE);

      /************************************************************
       *                                                          *
       *   This routine changes the output of the current element to *
       * the logic value defined as "VALUE" while returning to the  *
       * previous element.  If the value of "FLAG" is "1B" (one bit  *
       * binary 1), the routine also  assigns  the  element to the   *
       * proper test value.                                        *
       *                                                          *
       ************************************************************/
   DCL FLAG                 BIT(*),
       VALUE                BIT(*);

   IF FLAG THEN SELECT(VALUE);
    WHEN(CRITICAL_ZERO)
     TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
       | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRITO);
    WHEN(CRITICAL_ONE)
     TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
       | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRIT1);

    WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
     TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGH;

    OTHERWISE;
   END;

   IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH THEN
    TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
      | (TEST_FLAG(INLINE(ELEMENT_NO)) & HIGH_PATH);

   STACK_TOP = STACK_TOP - 1;
   PREDIR(STACK_TOP) = AHEAD;
   OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;

IF TRACE_FLAG THEN CALL TRACE('TGCONFR');

END CONFLECT_RETURN;

/**TGCPATH**/
FIND_CRITICAL_PATH: PROC     RETURNS( BIT(1) );

      /**************************************************************
       *                                                            *
       *   This routine defines the first critical path in the wGS  *
       * network.  If.an invalid path is found, the routine returns *
       * a value "1B" (one bit binary 1). Otherwise, the routine    *
       * returns a value "OB" to the calling program.               *
       *                                                            *
       **************************************************************/


   DCL LOGIC_VALUE          BIT(4)          INIT('0000'B),
       PFLAG                BIT(1)          INIT('0'B),
       BIFLAG               BIT(1)          INIT('0'B);

   PREDIR(STACK_TOP) = AHEAD;
   PREDIR(ZERO) = BACK;

   DO WHILE(STACK_TOP ¬= 0);
    ELEMENT_NO = STACK(STACK_TOP);
    SELECT(TYPE(ELEMENT_NO));


/* START        PROCESS POSITIVE SWITCH.
*/

      WHEN(POSITIVE_SWITCH) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (PS.AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE   */
        PREDIR(STACK_TOP) = INDIR;
        NEXT_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
        THEN DO;
         OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNCE;
        END;

        ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) ¬=
                                    OUT_VALUE(ELEMENT_NO,QSELECT)
        THEN RETURN(YES);

       END;

       WHEN(INDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (PS,INDIR)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */

        PREDIR(STACK_TOP) = CNTDIR;
        NEXT_ELMT = CNTLINE(ELEMENT_NO);

        IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
        THEN DO;
         OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNCE;
        END;

        ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT)¬= CRITICAL_ONE
         THEN STACK_TOP = STACK_TOP - 1;

        ELSE RETURN(YES);

       END;

       OTHERWISE
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;

      END;
     END;

/*END
*/

/*START       PROCESS NGETIVE SWITCH.
*/

     WHEN(NEGATIVE_SWITCH) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (NS,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */

        PREDIR(STACK_TOP) = INDIR;
        NEXT_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
        THEN DO;
         OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNCE;
        END;
```

```
         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) ¬=
                                  OUT_VALUE(ELEMENT_NO,QSELECT)
         THEN RETURN(YES);

        END;

        WHEN(INDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (NS,INDIR)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */

         PREDIR(STACK_TOP) = CNTDIR;
         NEXT_ELMT = CNTLINE(ELEMENT_NO);

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNCE;
         END;

         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
          THEN STACK_TOP = STACK_TOP - 1;

         ELSE RETURN(YES);

        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;
      END;

/*END
*/

/*START       PROCESS GENERAL WIRED-LOGIC GATE.
*/

      WHEN(GENERAL_WIRED) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (WG,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */
         IF OUT_VALUE(INLINE(ELEMENT_NO),QSELECT) =
                                    OUT_VALUE(ELEMENT_NO,QSELECT)
          | OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) =
                                    OUT_VALUE(ELEMENT_NO,QSELECT)
         THEN STACK_TOP = STACK_TOP - 1;

         ELSE DO;
          PREDIR(STACK_TOP) = INDIR;
          NEXT_ELMT = INLINE(ELEMENT_NO);
          OTHER_ELMT = CNTLINE(ELEMENT_NO);
          IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
           & (TEST_FLAG(NEXT_ELMT) & CRITO) = CRITO
           & (TEST_FLAG(OTHER_ELMT) & CRITO) = NO_TEST
          THEN DO;
           NEXT_ELMT = OTHER_ELMT;
           PREDIR(STACK_TOP) = CNTDIR;
          END;

          ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
          & (TEST_FLAG(NEXT_ELMT) & CRIT1) = CRIT1
          & (TEST_FLAG(OTHER_ELMT) & CRIT1) = NO_TEST
         THEN DO;
          NEXT_ELMT = OTHER_ELMT;
          PREDIR(STACK_TOP) = CNTDIR;
         END;

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
 %INCLUDE TGCNCE;
         END;
         ELSE RETURN(YES);

        END;
       END;

       OTHERWISE
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;

      END;
     END;

/*END
*/

/*START      PROCESS PULL-UP/DOWN WIRED-LOGIC.
*/

     WHEN(PULL_UD_WIRED) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (WP,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */
        PREDIR(STACK_TOP) = INDIR;
        NEXT_ELMT = INLINE(ELEMENT_NO);
        IF CNTLINE(ELEMENT_NO) = WEAK_ONE THEN DO;
          /* PULL_UP LOAD */
         IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
         THEN STACK_TOP = STACK_TOP - 1;

         ELSE DO;
          IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
          THEN DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
 %INCLUDE TGCNCE;
          END;

          ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
          THEN STACK_TOP = STACK_TOP - 1;

          ELSE RETURN(YES);
         END;

        END;

        ELSE DO;
          /* PULL-DOWN LOAD */
         IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
         THEN STACK_TOP = STACK_TOP - 1;

         ELSE DO;
```

```
           IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
           THEN DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNCE;
           END;

           ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
           THEN STACK_TOP = STACK_TOP - 1;

           ELSE RETURN(YES);
          END;

         END;

        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;
      END;

/*END
*/

/*START      PROCESS COMPLIMENTARY WIRED-LOGIC GATE.
*/

      WHEN(COMPLIMENTARY_WIRED) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (WC,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */
         IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
         THEN DO;
          NEXT_ELMT = INLINE(ELEMENT_NO);
          PREDIR(STACK_TOP) = INDIR;
         END;

         ELSE DO;
          NEXT_ELMT = CNTLINE(ELEMENT_NO);
          PREDIR(STACK_TOP) = CNTDIR;
         END;

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNCE;
         END;

         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) =
                                  OUT_VALUE(ELEMENT_NO,QSELECT)
         THEN STACK_TOP = STACK_TOP - 1;

         ELSE RETURN(YES);

        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;
       END;
      END;

/*END
*/

/*START      PROCESS HIGH WIRED-LOGIC GATE.
*/

      WHEN(HIGH_WIRED) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (WH,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */
         IF OUT_VALUE(INLINE(ELEMENT_NO),QSELECT) =
                                OUT_VALUE(ELEMENT_NO,QSELECT)
          | OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) =
                                OUT_VALUE(ELEMENT_NO,QSELECT)
         THEN STACK_TOP = STACK_TOP - 1;

         ELSE DO;
          NEXT_ELMT = INLINE(ELEMENT_NO);
          OTHER_ELMT = CNTLINE(ELEMENT_NO);

          IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
           & (TEST_FLAG(NEXT_ELMT) & CRIT1) = CRIT1
           & (TEST_FLAG(OTHER_ELMT) & CRIT1) = NO_TEST
          THEN DO;
           NEXT_ELMT = OTHER_ELMT;
           PREDIR(STACK_TOP) = CNTDIR;
          END;
          ELSE PREDIR(STACK_TOP) = INDIR;

          IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
          THEN DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
 %INCLUDE TGCNCE;
          END;
          ELSE RETURN(YES);

         END;
        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;
      END;

/*END
*/

/*START      PROCESS LOW WIRED-LOGIC GATE.
*/

      WHEN(LOW_WIRED) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (WL,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */
         IF OUT_VALUE(INLINE(ELEMENT_NO),QSELECT) =
                                        OUT_VALUE(ELEMENT_NO,QSELECT)
         | OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) =
                                        OUT_VALUE(ELEMENT_NO,QSELECT)
        THEN STACK_TOP = STACK_TOP - 1;

        ELSE DO;
         NEXT_ELMT = INLINE(ELEMENT_NO);
         OTHER_ELMT = CNTLINE(ELEMENT_NO);

         IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
          & (TEST_FLAG(NEXT_ELMT) & CRITO) = CRITO
          & (TEST_FLAG(OTHER_ELMT) & CRITO) = NO_TEST
          THEN DO;
           NEXT_ELMT = OTHER_ELMT;
           PREDIR(STACK_TOP) = CNTDIR;
          END;
          ELSE PREDIR(STACK_TOP) = INDIR;

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNCE;
         END;
         ELSE RETURN(YES);

        END;
       END;

       OTHERWISE
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;

      END;
     END;

/*END
*/

/* START        PROCESS INPUT POSITIVE SWITCH.
*/

     WHEN(INPUT_POSITIVE) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (IP,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */

        PREDIR(STACK_TOP) = CNTDIR;
        NEXT_ELMT = CNTLINE(ELEMENT_NO);

        IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
        THEN DO;
         OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNCE;
        END;

        ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
         THEN STACK_TOP = STACK_TOP - 1;

        ELSE RETURN(YES);

       END;
```

```
        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;
      END;

/*END
*/

/* START        PROCESS INPUT NEGATIVE SWITCH.
*/

      WHEN(INPUT_NEGATIVE) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (IN,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */

         PREDIR(STACK_TOP) = CNTDIR;
         NEXT_ELMT = CNTLINE(ELEMENT_NO);

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNCE;
         END;

         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
          THEN STACK_TOP = STACK_TOP - 1;

         ELSE RETURN(YES);

        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;
      END;

/*END
*/

/*START    PROCESS PRIMITIVE-INPUT ELEMENT TO.
*/
      WHEN(PRIMITIVE_INPUT) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (PI)');
          /* PROCESS RETURN FROM THIS ELEMENT */
       STACK_TOP = STACK_TOP - 1;
      END;

/*END
*/

/* START        PROCESS BIPOSITIVE SWITCH.
*/

      WHEN(BIPOSITIVE_SWITCH) DO;

IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,IN)');
       SELECT(PREDIR(STACK_TOP));
      WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,AHEAD)');
        /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */
       IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT THEN DO;
        IF OUT_VALUE(ELEMENT_NO-1,QSELECT) = DONT_CARE
        THEN DO;
         PFLAG,BIFLAG = YES;
         CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO - 1);
        END;
        ELSE PFLAG = NO;
       END;
       ELSE IF OUT_VALUE(ELEMENT_NO+1,QSELECT) = DONT_CARE
       THEN PFLAG = YES;
       ELSE PFLAG = NO;

       IF PFLAG THEN DO;
        PREDIR(STACK_TOP) = INDIR;
        NEXT_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
        THEN DO;
         OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
 %INCLUDE TGCNCE;
        END;

        ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) ¬=
                                    OUT_VALUE(ELEMENT_NO,QSELECT)
        THEN RETURN(YES);

        IF BIFLAG THEN DO;
         BIFLAG = NO;
         CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
        END;

       END;

       ELSE DO;
  IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,OUT)');
        STACK_TOP = STACK_TOP - 1;
        IF PREDIR(STACK_TOP) = INDIR
        THEN NEXT_ELMT = CNTLINE(STACK(STACK_TOP));
        ELSE NEXT_ELMT = INLINE(STACK(STACK_TOP));
        OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
        OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE;
  %INCLUDE TGCNCE;
       END;

      END;

      WHEN(INDIR) DO;

       IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT THEN DO;
        BIFLAG = YES;
        CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO - 1);
       END;

IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,INDIR)');
        /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */
       PREDIR(STACK_TOP) = CNTDIR;
       NEXT_ELMT = CNTLINE(ELEMENT_NO);

       IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
       THEN DO;
```

```
         OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNCE;
        END;

        ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
         THEN STACK_TOP = STACK_TOP - 1;

        ELSE RETURN(YES);

        IF BIFLAG THEN DO;
         BIFLAG = NO;
         CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
        END;

       END;

       OTHERWISE
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;

      END;

     END;

/*END
*/

/*START       PROCESS BINGETIVE SWITCH.
*/

     WHEN(BINEGATIVE_SWITCH) DO;

IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,IN)');
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BN,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */

        IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT THEN DO;
         IF OUT_VALUE(ELEMENT_NO-1,QSELECT) = DONT_CARE
         THEN DO;
          PFLAG,BIFLAG = YES;
          CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO - 1);
         END;
         ELSE PFLAG = NO;
        END;
        ELSE IF OUT_VALUE(ELEMENT_NO+1,QSELECT) = DONT_CARE
        THEN PFLAG = YES;
        ELSE PFLAG = NO;

        IF PFLAG THEN DO;
         PREDIR(STACK_TOP) = INDIR;
         NEXT_ELMT = INLINE(ELEMENT_NO);

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
 %INCLUDE TGCNCE;
         END;

         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) ¬=
                                     OUT_VALUE(ELEMENT_NO,QSELECT)
         THEN RETURN(YES);
```

```
          IF BIFLAG THEN DO;
           BIFLAG = NO;
           CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
          END;

         END;

         ELSE DO;
   IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BP,OUT)');
          STACK_TOP = STACK_TOP - 1;
          IF PREDIR(STACK_TOP) = INDIR
          THEN NEXT_ELMT = CNTLINE(STACK(STACK_TOP));
          ELSE NEXT_ELMT = INLINE(STACK(STACK_TOP));
          OUT_VALUE(NEXT_ELMT,QSELECT) = OUT_VALUE(ELEMENT_NO,QSELECT);
          OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE;
   %INCLUDE TGCNCE;
         END;

        END;

        WHEN(INDIR) DO;

         IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT THEN DO;
          BIFLAG = YES;
          CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO - 1);
         END;

 IF TRACE_FLAG THEN CALL TRACE('TGFCPATH (BN,INDIR)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE */
         PREDIR(STACK_TOP) = CNTDIR;
         NEXT_ELMT = CNTLINE(ELEMENT_NO);

         IF OUT_VALUE(NEXT_ELMT,QSELECT) = DONT_CARE
         THEN DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
 %INCLUDE TGCNCE;
         END;

         ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
          THEN STACK_TOP = STACK_TOP - 1;

         ELSE RETURN(YES);

         IF BIFLAG THEN DO;
          BIFLAG = NO;
          CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
         END;

        END;

        OTHERWISE
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;

       END;

      END;

 /*END
 */


          /* ELEMENT NOT FOUNT.                        */
      OTHERWISE DO;
       PUT FILE(SYSPRINT) SKIP EDIT('ELEMENT NOT FOUND')(A);
       RETURN(YES);
      END;

     END;
    END;

    IF PREDIR(STACK_TOP) = AHEAD THEN RETURN(YES);
    ELSE RETURN(NO);

END FIND_CRITICAL_PATH;

/**TGFPFIN**/
FIND_PATH_FOR_INIT_NODE: PROC(QTOP) RETURNS( BIT(1) );

      /**************************************************************
       *                                                            *
       *   This routine generates a pattern for pre-initializing the *
       * output logic value of the element as "QTOP".  If an invalid *
       * path is found, the routine returns a value "1B" (one bit    *
       * binary 1). Otherwise, the routine returns a value "OB" to   *
       * calling program.                                           *
       *                                                            *
       **************************************************************/


    DCL QTOP                  FIXED BIN(31),
        SWAP                  FIXED BIN(31)  INIT(0),
        I                     FIXED BIN(31)  INIT(0);

IF TRACE_FLAG THEN CALL TRACE('TGFPFIN');

    DO I = ONE TO TEMP_QTOP;
       OUT_VALUE(QUEUE(I,TWO),TWO) = DONT_CARE;
    END;

    IF OUT_VALUE(QTOP,ONE) = CRITICAL_ZERO
    THEN
       OUT_VALUE(QTOP,TWO) = LOGIC_ONE;
    ELSE
       OUT_VALUE(QTOP,TWO) = LOGIC_ZERO;
    QSELECT = TWO;
    SWAP = QUEUE_TOP;
    QUEUE_TOP,STACK_TOP = ONE;
    STACK(STACK_TOP),QUEUE(QUEUE_TOP,QSELECT) = QTOP;

    IF JUSTIFY_OTHER_LINE THEN RETURN(YES);
    CALL GENERATE_TEST(ZERO);

    TEMP_QTOP = QUEUE_TOP;
    QUEUE_TOP = SWAP;
    QSELECT = ONE;

END FIND_PATH_FOR_INIT_NODE;

/**TGGTEST**/
GENERATE_TEST: PROC(FLAG);

      /**************************************************************
       *                                                            *
       *   This routine processes a test as a record, opens the file *
       * "OUT", writes the record in the file "OUT", then closes the *
    * file "OUT".  A value of "0" for "FLAG" indicates that the   *
    * test is for the pre-initialization process. A value of "2"  *
    * for "FLAG" indicates that the current test must be used     *
    * before or after the previous test.  Any other value         *
    * indicates a test with no condition.                         *
    *                                                             *
    ***************************************************************/

   DCL FLAG                    FIXED BIN(31),
       BEFOR_NO                PIC 'ZZZZZZZ9' INIT(0),
       TEMP                    PIC 'ZZZZZZZ9' INIT(0),
       TEXT                    CHAR(121) VAR  INIT(' '),
       (L,K)                   FIXED BIN(31);

IF TRACE_FLAG THEN CALL TRACE('TGGTEST');

   IF FLAG = ZERO THEN DO;
      TEMP = TEST_NO + 1;
      TEXT = BEFOR_NO || ' ' || TEMP;
   END;

   ELSE DO;
      IF FLAG = TWO THEN BEFOR_NO = TEST_NO;
      TEST_NO = TEST_NO + 1;
      TEXT = TEST_NO || ' ' || BEFOR_NO;
   END;

   TEMP = OP->LINEOUT_NO;

   IF FLAG = ZERO
   THEN TEXT = TEXT || ' ' || TEMP || ' ' || 'X' || ' ';

   ELSE IF OUT_VALUE(TEMP,QSELECT) = CRITICAL_ZERO
   THEN TEXT = TEXT || ' ' || TEMP || ' ' || '0' || ' ';
   ELSE TEXT = TEXT || ' ' || TEMP || ' ' || '1' || ' ';

   IP = INPUTS_HEAD->NEXT_IP;
   K = 29;

   DO L = K+1 TO 121 WHILE(IP ¬= NULL);

    SELECT(OUT_VALUE(IP->LINEIN_NO,QSELECT));
     WHEN(CRITICAL_ZERO,LOGIC_ZERO)
      TEXT = TEXT || '0';
     WHEN(CRITICAL_ONE,LOGIC_ONE)
      TEXT = TEXT || '1';
     OTHERWISE
      TEXT = TEXT || 'X';
    END;
    IP = IP->NEXT_IP;
   END;

   OPEN FILE(OUT) OUTPUT;
   WRITE FILE(OUT) FROM(TEXT);
   CLOSE FILE(OUT);

IF TRACE_FLAG & QSELECT = ONE THEN DO;
   DO I = 1 TO QUEUE_TOP;
      PUT FILE(FTRACE) SKIP EDIT('ELMT# = ',QUEUE(I,ONE),'  TEST = ',
       TEST_FLAG(QUEUE(I,ONE)))(COL(2),A,P'ZZZZZZZ9',A,P'ZZZ9');
   END;
END;
END GENERATE_TEST;
```

```
/**TGINPUT**/
INPUT_DATA: PROC;

      /***************************************************************
       *                                                             *
       *   This routine reads a wGS network into the program.        *
       *                                                             *
       ***************************************************************/


   DCL START                 POINTER,
       PRIV                  POINTER,
       P                     POINTER,
       SWAP                  POINTER;

IF TRACE_FLAG THEN CALL TRACE('TGINPUT');

   COMMAND = '  ';

   DO WHILE(¬EOF);
      READ FILE(IN) INTO(TEXT);
      COMMAND = SUBSTR(TEXT,1,2);
      TEXT = SUBSTR(TEXT,4) || ' ';

      SELECT(COMMAND);

            /* ADD A POSITIVE SWITCH TO THE STRUCTURE.  */

         WHEN('PS','ps','Ps','pS') DO;
            GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
            INLINE(ELEMENT_NO) = IN_NO;
            CNTLINE(ELEMENT_NO) = CNT_NO;
            TYPE(ELEMENT_NO) = POSITIVE_SWITCH;
            TEST_FLAG(ELEMENT_NO) = NO_INIT;
         END;


            /* ADD A NEGATIVE SWITCH TO THE STRUCTURE.  */

         WHEN('NS','ns','Ns','nS') DO;
            GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
            INLINE(ELEMENT_NO) = IN_NO;
            CNTLINE(ELEMENT_NO) = CNT_NO;
            TYPE(ELEMENT_NO) = NEGATIVE_SWITCH;
            TEST_FLAG(ELEMENT_NO) = NO_INIT;
         END;


            /* ADD A GENERAL WIRED-LOGIC TO THE STRUCTURE.  */

         WHEN('WG','wg','Wg','wG') DO;
            GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
            INLINE(ELEMENT_NO) = IN_NO;
            CNTLINE(ELEMENT_NO) = CNT_NO;
            TYPE(ELEMENT_NO) = GENERAL_WIRED;
            TEST_FLAG(ELEMENT_NO) = NO_INIT;
         END;


            /* ADD A PULL-UP WIRED-LOGIC TO THE STRUCTURE. */

         WHEN('WU','wu','Wu','wU') DO;
            GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   TYPE(ELEMENT_NO) = PULL_UD_WIRED;
   CNTLINE(ELEMENT_NO) = WEAK_ONE;
   TEST_FLAG(ELEMENT_NO) = WU_INIT;
END;


   /* ADD A COMPLIMENTARY WIRED-LOGIC TO THE STRUCTURE. */

WHEN('WC','wc','Wc','wC') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = COMPLIMENTARY_WIRED;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD A HIGH WIRED-LOGIC TO THE STRUCTURE.      */

WHEN('WH','wh','Wh','wH') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = HIGH_WIRED;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD A LOW WIRED-LOGIC TO THE STRUCTURE.      */

WHEN('WL','wl','Wl','wL') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = LOW_WIRED;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD AN INPUT POSITIVE SWITCH TO THE STRUCTURE. */

WHEN('IP','ip','Ip','iP') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = INPUT_POSITIVE;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD AN INPUT NEGATIVE SWITCH TO THE STRUCTURE. */

WHEN('IN','in','In','iN') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = INPUT_NEGATIVE;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD AN PRIMITIVE-INPUT ELEMENT TO THE STRUCTURE. */
```

```
WHEN('PI','pi','Pi','pI') DO;
   NO_INPUTS = NO_INPUTS + 1;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   TYPE(ELEMENT_NO) = PRIMITIVE_INPUT;
   CNTLINE(ELEMENT_NO) = VDD;
   ALLOCATE PRIMITIVE_INPUTS SET(IP->NEXT_IP);
   IP = IP->NEXT_IP;
   IP->LINEIN_NO = ELEMENT_NO;
   TEST_FLAG(ELEMENT_NO) = ALL_INIT;
END;


   /* ADD A BIPOSITIVE SWITCH TO THE STRUCTURE. */

WHEN('BP','bp','Bp','bP') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = BIPOSITIVE_SWITCH;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD A BINEGATIVE SWITCH TO THE STRUCTURE. */

WHEN('BN','bn','Bn','bN') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO,CNT_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   CNTLINE(ELEMENT_NO) = CNT_NO;
   TYPE(ELEMENT_NO) = BINEGATIVE_SWITCH;
   TEST_FLAG(ELEMENT_NO) = NO_INIT;
END;


   /* ADD A PULL-DOWN WIRED-LOGIC TO THE STRUCTURE. */

WHEN('WD','wd','Wd','wD') DO;
   GET STRING(TEXT) LIST(ELEMENT_NO,IN_NO);
   INLINE(ELEMENT_NO) = IN_NO;
   TYPE(ELEMENT_NO) = PULL_UD_WIRED;
   CNTLINE(ELEMENT_NO) = WEAK_ZERO;
   TEST_FLAG(ELEMENT_NO) = WD_INIT;
END;


   /* READ PRIMITIVE OUTPUT LINES INTO THE PROGRAM. */

WHEN('PO','po','Po','pO') DO;
   GET STRING(TEXT) LIST(NO_OUTPUTS);

   DO I = 1 TO NO_OUTPUTS;

      DO WHILE(SUBSTR(TEXT,1,1) = ' ');
         TEXT = SUBSTR(TEXT,2);
      END;

      K = INDEX(TEXT,' ');
      TEXT = SUBSTR(TEXT,K);
      GET STRING(TEXT) LIST(ELEMENT_NO);
      ALLOCATE PRIMITIVE_OUTPUTS SET(OP->NEXT_OP);
      OP = OP->NEXT_OP;
      OP->LINEOUT_NO = ELEMENT_NO;
   END;
```

```
         END;


            /* END OF FILE REACHED, EXIT THIS ROUTINE. */

         WHEN('EO','eo','Eo','eO') DO;
            EOF = YES;
         END;

             /* COMMAND NOT FOUNT, PROBABLY COMMENTS. */

         OTHERWISE;

      END;

      COMMAND = '  ';
   END;

   IP->NEXT_IP = NULL;
   OP->NEXT_OP = NULL;

/* SORT INPUT LINES */

   START = INPUTS_HEAD;
   DO WHILE(START->NEXT_IP ¬= NULL);
      SWAP = START;
      P,PRIV = START->NEXT_IP;
      IP = P->NEXT_IP;
      DO WHILE(IP ¬= NULL);
         IF INLINE(IP->LINEIN_NO) < INLINE(P->LINEIN_NO)
         THEN DO;
            SWAP = PRIV;
            P = IP;
         END;
         PRIV = IP;
         IP = IP->NEXT_IP;
      END;
      IF START ¬= SWAP THEN DO;
         SWAP->NEXT_IP = P->NEXT_IP;
         P->NEXT_IP = START->NEXT_IP;
         START->NEXT_IP,START = P;
      END;
      ELSE START = P;
   END;

/* END SORT INPUT LINES */

END INPUT_DATA;

/**TGISCFR**/
CONFLECT_IS_RETURN: PROC(VALUE);

      /***************************************************************
       *                                                             *
       *   This routine changes the output of the current element to *
       * the logic value defined as "VALUE" while returning to the   *
       * previous element.                                           *
       *                                                             *
       ***************************************************************/


   DCL VALUE                  BIT(*);
   STACK_TOP = STACK_TOP - 1;
   PREDIR(STACK_TOP) = AHEAD;
   OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;

IF TRACE_FLAG THEN CALL TRACE('TGISCFR');

END CONFLECT_IS_RETURN;

/**TGJOPATH**/
JUSTIFY_OTHER_LINE: PROC     RETURNS( BIT(1) );

      /****************************************************************
       *                                                              *
       *   This routine will justify other lines in the network      *
       * while defining other critical paths as conflect processes   *
       * and lines with the value "critical High-Impedance." If an   *
       * invalid path is found, the routine returns a value "1B"     *
       * (one bit binary 1),  otherwise the routine returns a value  *
       * "0B" to the calling program.                                 *
       *                                                              *
       ****************************************************************/


   DCL LOGIC_VALUE          BIT(4)          INIT('0000'B),
       PATH_SET             BIT(4)          INIT('0000'B),
       TEST_SET             BIT(4)          INIT('0000'B),
       FIRST_BI_INUSE       FIXED BIN(31)   INIT(-1),
       SECOND_BI_INUSE      FIXED BIN(31)   INIT(-2),
       PFLAG                BIT(1)          INIT('0'B),
       BIFLAG               BIT(1)          INIT('0'B);

   PREDIR(ZERO) = BACK;
   PREDIR(STACK_TOP) = AHEAD;

   DO WHILE(STACK_TOP ¬= 0);
    ELEMENT_NO = STACK(STACK_TOP);

    SELECT(TYPE(ELEMENT_NO));

     WHEN(POSITIVE_SWITCH) DO;
%INCLUDE TGPS;
     END;

     WHEN(NEGATIVE_SWITCH) DO;
%INCLUDE TGNS;
     END;

%INCLUDE TGWG;

%INCLUDE TGWP;

%INCLUDE TGWC;

%INCLUDE TGWH;

%INCLUDE TGWL;

%INCLUDE TGIP;

%INCLUDE TGIN;

/*START    PROCESS PRIMITIVE-INPUT ELEMENT.
*/
     WHEN(PRIMITIVE_INPUT) DO;
```

```
      STACK_TOP = STACK_TOP - 1;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (PI,RETURN)');
      /* PROCESS RETURN FROM THIS ELEMENT */
     END;

/*END
*/

/*START    PROCESS BIPOSITIVE SWITCH.
*/
     WHEN(BIPOSITIVE_SWITCH) DO;

IF TRACE_FLAG THEN CALL TRACE('TGJOPATH(BIPOSITIVE)');
      SELECT(CNTLINE(ELEMENT_NO)):
       WHEN(BISWITCH_CNT) DO;
        PFLAG = YES;
        CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO-1);
        CNTLINE(ELEMENT_NO-1) = FIRST_BI_INUSE;
       END;
       WHEN(FIRST_BI_INUSE,SECOND_BI_INUSE) DO;
         /* PROCESS RETURN FROM THIS ELEMENT */
        PFLAG = NO;
        STACK_TOP = STACK_TOP - 1;
        PREDIR(STACK_TOP) = AHEAD;
        SELECT(OUT_VALUE(STACK(STACK_TOP),QSELECT));
         WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
          OUT_VALUE(ELEMENT_NO,QSELECT) =
                                OUT_VALUE(STACK(STACK_TOP),QSELECT);
         OTHERWISE
          OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE;
        END;
       END;
       OTHERWISE DO;
        PFLAG = YES;
        IF CNTLINE(ELEMENT_NO+1) = BISWITCH_CNT
        THEN CNTLINE(ELEMENT_NO+1) = SECOND_BI_INUSE;
       END;
      END;

      IF PFLAG THEN DO;
       IF PREDIR(STACK_TOP) = AHEAD
       THEN STACK(STACK_TOP+1) = CNTLINE(ELEMENT_NO);
       ELSE STACK(STACK_TOP+1) = INLINE(ELEMENT_NO);
%INCLUDE TGPS;
       IF STACK(STACK_TOP+1) = ELEMENT_NO THEN DO;
        SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
         WHEN(CRITICAL_ZERO) DO;
          TEST_SET = NO_TEST;  PATH_SET = CRITO;
         END;
         WHEN(CRITICAL_ONE) DO;
          TEST_SET = NO_TEST;  PATH_SET = CRIT1;
         END;
         WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
          IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ZERO
          THEN DO;  TEST_SET = HIGH;   PATH_SET = HIGH_PATH;   END;
          ELSE DO;  TEST_SET = NO_TEST;  PATH_SET = NO_TEST;   END;
         END;
         OTHERWISE DO;
           TEST_SET = NO_TEST;  PATH_SET = NO_TEST;
         END;
        END;

        IF CNTLINE(ELEMENT_NO+1) = SECOND_BI_INUSE
        THEN CNTLINE(ELEMENT_NO+1) = BISWITCH_CNT;
```

```
         ELSE DO;
          CNTLINE(ELEMENT_NO-1) = CNTLINE(ELEMENT_NO);
          CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
         END;

         TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) |
          TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO)) & PATH_SET);

         IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT
         THEN TEST_FLAG(ELEMENT_NO-1) = TEST_FLAG(ELEMENT_NO-1)
           | TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO-1)) & PATH_SET);

         ELSE TEST_FLAG(ELEMENT_NO+1) = TEST_FLAG(ELEMENT_NO+1)
           | TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO+1)) & PATH_SET);

        END;
       END;

      END;

/*END
*/

/*START    PROCESS BINEGATIVE SWITCH.
*/
     WHEN(BINEGATIVE_SWITCH) DO;

IF TRACE_FLAG THEN CALL TRACE('TGJOPATH(BINEGATIVE)');
      SELECT(CNTLINE(ELEMENT_NO));
       WHEN(BISWITCH_CNT) DO;
        PFLAG = YES;
        CNTLINE(ELEMENT_NO) = CNTLINE(ELEMENT_NO-1);
        CNTLINE(ELEMENT_NO-1) = FIRST_BI_INUSE;
       END;
       WHEN(FIRST_BI_INUSE,SECOND_BI_INUSE) DO;
         /* PROCESS RETURN FROM THIS ELEMENT */
        PFLAG = NO;
        STACK_TOP = STACK_TOP - 1;
        PREDIR(STACK_TOP) = AHEAD;
        SELECT(OUT_VALUE(STACK(STACK_TOP),QSELECT));
         WHEN(HIGH0_IMPEDANCE,HIGH1_IMPEDANCE)
          OUT_VALUE(ELEMENT_NO,QSELECT) =
                                OUT_VALUE(STACK(STACK_TOP),QSELECT);
         OTHERWISE
          OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE;
        END;
       END;
       OTHERWISE DO;
        PFLAG = YES;
        IF CNTLINE(ELEMENT_NO+1) = BISWITCH_CNT
        THEN CNTLINE(ELEMENT_NO+1) = SECOND_BI_INUSE;
       END;
      END;

      IF PFLAG THEN DO;
       IF PREDIR(STACK_TOP) = AHEAD
       THEN STACK(STACK_TOP+1) = CNTLINE(ELEMENT_NO);
       ELSE STACK(STACK_TOP+1) = INLINE(ELEMENT_NO);
%INCLUDE TGNS;
       IF STACK(STACK_TOP+1) = ELEMENT_NO THEN DO;
        SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
         WHEN(CRITICAL_ZERO) DO;
          TEST_SET = NO_TEST;  PATH_SET = CRIT0;
         END;
```

```
          WHEN(CRITICAL_ONE) DO;
           TEST_SET = NO_TEST;  PATH_SET = CRIT1;
          END;
          WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
           IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ONE
           THEN DO;  TEST_SET = HIGH;   PATH_SET = HIGH_PATH;   END;
           ELSE DO;  TEST_SET = NO_TEST;  PATH_SET = NO_TEST;   END;
          END;
          OTHERWISE DO;
            TEST_SET = NO_TEST;  PATH_SET = NO_TEST;
          END;
         END;

         IF CNTLINE(ELEMENT_NO+1) = SECOND_BI_INUSE
         THEN CNTLINE(ELEMENT_NO+1) = BISWITCH_CNT;
         ELSE DO;
          CNTLINE(ELEMENT_NO-1) = CNTLINE(ELEMENT_NO);
          CNTLINE(ELEMENT_NO) = BISWITCH_CNT;
         END;

         TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) |
          TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO)) & PATH_SET);

         IF CNTLINE(ELEMENT_NO) = BISWITCH_CNT
         THEN TEST_FLAG(ELEMENT_NO-1) = TEST_FLAG(ELEMENT_NO-1)
          | TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO-1)) & PATH_SET);

         ELSE TEST_FLAG(ELEMENT_NO+1) = TEST_FLAG(ELEMENT_NO+1)
          | TEST_SET | (TEST_FLAG(INLINE(ELEMENT_NO+1)) & PATH_SET);

        END;
       END;

      END;

/*END
*/

          /* ELEMENT NOT FOUNT.                      */

      OTHERWISE DO;
       PUT FILE(SYSPRINT) SKIP EDIT('ELEMENT NOT FOUND')(A);
       RETURN(YES);
      END;

     END;
    END;

    IF PREDIR(STACK_TOP) = AHEAD THEN RETURN(YES);
    ELSE RETURN(NO);

END JUSTIFY_OTHER_LINE;

/**TGNETWRK**/
PROCESS_NETWORK: PROC       RETURNS( BIT(1) );

      /*****************************************************************
       *                                                               *
       *   This routine processes the CPTGTL algorithm for test        *
       * generation purposes.  The sensitization and justification     *
       * process starts  from this program.  Generating the actual     *
       * tests and  pre-initialization  patterns  are  controlled by   *
       * this  routine.   Printing  the  generated  tests  are  also   *
       * controlled  by this  program.  If an invalid path is found, *
       * the  routine  returns  a  value  "1B"  (one bit binary),     *
       * otherwise the routine  returns a value "0B" to the calling  *
       * program.                                                     *
       *                                                              *
       ***************************************************************/


    DCL CRIT_VALUE            BIT(4)          INIT('0000'B),
        PATH_SET              BIT(4)          INIT('0000'B),
        QTOP                  FIXED BIN(31)   INIT(0),
        ELMT_NO               FIXED BIN(31)   INIT(0),
        INIT_FLAG             BIT(1)          INIT('0'B),
        INTR_FLAG             BIT(1)          INIT('0'B),
        PRE_INIT              BIT(1)          INIT('0'B),
        PFLAG                 BIT(1)          INIT('0'B),
        SET_VALUE             FIXED BIN(31)   INIT(1),
        PRIM_QTOP             FIXED BIN(31)   INIT(0),
        I                     FIXED BIN(31)   INIT(0);

 IF TRACE_FLAG THEN CALL TRACE('TGNETWRK');

    DO WHILE(OUTPUTS_HEAD->NEXT_OP ¬= NULL);
       PRIV_OP = OUTPUTS_HEAD;
       OP = PRIV_OP->NEXT_OP;
       CRIT_VALUE = CRITICAL_ZERO;
       SET_VALUE = ONE;

       DO WHILE(OP ¬= NULL);

        DO I = ONE TO QUEUE_TOP;
           OUT_VALUE(QUEUE(I,QSELECT),QSELECT) = DONT_CARE;
        END;

        IF SET_VALUE = ONE & CRIT_VALUE = CRITICAL_ZERO
         & (TEST_FLAG(OP->LINEOUT_NO) & (CRITO | HIGHP))
         = (CRITO | HIGHP) THEN CRIT_VALUE = CRITICAL_ONE;

        STACK_TOP,QUEUE_TOP = ONE;
        STACK(STACK_TOP),QUEUE(QUEUE_TOP,QSELECT) = OP->LINEOUT_NO;
        OUT_VALUE(STACK(STACK_TOP),QSELECT) = CRIT_VALUE;

        IF FIND_CRITICAL_PATH THEN RETURN(YES);
        QTOP = QUEUE_TOP;

        DO I = QTOP TO ONE BY -1;
         ELMT_NO = QUEUE(I,QSELECT);
         SELECT(TYPE(ELMT_NO));
             /* PROCESS GENERAL WIRED-LOGIC.           */

          WHEN(GENERAL_WIRED) DO;
           IF INIT_FLAG THEN INTR_FLAG = YES;
%INCLUDE TGCJOP;
          END;


             /* PROCESS COMPLIMENTARY WIRED-LOGIC.     */

          WHEN(COMPLIMENTARY_WIRED) DO;
           IF INTR_FLAG THEN DO;
            IF FIND_PATH_FOR_INIT_NODE(ELMT_NO) THEN RETURN(YES);
            PRE_INIT = YES;
            INTR_FLAG = NO;
           END;
```

```
%INCLUDE TGCJOP;
         END;


            /* PROCESS HIGH WIRED-LOGIC.                  */

         WHEN(HIGH_WIRED) DO;
          IF INTR_FLAG & OUT_VALUE(ELMT_NO,QSELECT) = CRITICAL_ONE
          THEN DO;
           IF FIND_PATH_FOR_INIT_NODE(ELMT_NO) THEN RETURN(YES);
           PRE_INIT = YES;
           INTR_FLAG = NO;
          END;
%INCLUDE TGCJOP;
         END;


            /* PROCESS LOW WIRED-LOGIC.                   */

         WHEN(LOW_WIRED) DO;
          IF INTR_FLAG & OUT_VALUE(ELMT_NO,QSELECT) = CRITICAL_ZERO
          THEN DO;
           IF FIND_PATH_FOR_INIT_NODE(ELMT_NO) THEN RETURN(YES);
           PRE_INIT = YES;
           INTR_FLAG = NO;
          END;
%INCLUDE TGCJOP;
         END;


            /* PROCESS INPUT_POSITIVE/NEGATIVE.           */

         WHEN(INPUT_POSITIVE,INPUT_NEGATIVE) DO;
          IF   INLINE(ELMT_NO) = ZERO
          THEN TEST_FLAG(ELMT_NO) = TEST_FLAG(ELMT_NO) | CRITO;
          ELSE TEST_FLAG(ELMT_NO) = TEST_FLAG(ELMT_NO) | CRIT1;
          PRIM_QTOP = 0;
          INIT_FLAG = YES;
         END;


            /* PROCESS POSITIVE/NEGATIVE SWITCH.          */

         WHEN(POSITIVE_SWITCH,NEGATIVE_SWITCH) DO;
          IF OUT_VALUE(ELMT_NO,QSELECT) = CRITICAL_ZERO THEN
             TEST_FLAG(ELMT_NO) = TEST_FLAG(ELMT_NO)
             | (TEST_FLAG(INLINE(ELMT_NO)) & CRITO);
          ELSE
             TEST_FLAG(ELMT_NO) = TEST_FLAG(ELMT_NO)
             | (TEST_FLAG(INLINE(ELMT_NO)) & CRIT1);
         END;


            /* PROCESS BIPOSITIVE/BINEGATIVE SWITCH.  */

         WHEN(BIPOSITIVE_SWITCH,BINEGATIVE_SWITCH) DO;
          IF OUT_VALUE(ELMT_NO,QSELECT) = CRITICAL_ZERO
          THEN PATH_SET = CRITO;
          ELSE IF OUT_VALUE(ELMT_NO,QSELECT) = CRITICAL_ONE
          THEN PATH_SET = CRIT1;
          ELSE PATH_SET = NO_TEST;
          TEST_FLAG(ELMT_NO) = TEST_FLAG(ELMT_NO)
             | (TEST_FLAG(INLINE(ELMT_NO)) & PATH_SET);
          IF CNTLINE(ELMT_NO) = BISWITCH_CNT
```

```
             THEN TEST_FLAG(ELMT_NO - 1) = TEST_FLAG(ELMT_NO - 1)
                  | (TEST_FLAG(INLINE(ELMT_NO - 1)) & PATH_SET);
             ELSE TEST_FLAG(ELMT_NO + 1) = TEST_FLAG(ELMT_NO + 1)
                  | (TEST_FLAG(INLINE(ELMT_NO + 1)) & PATH_SET);
            END;


               /* PROCESS PRIMITIVE INPUT                    */

            WHEN(PRIMITIVE_INPUT) DO;
             PRIM_QTOP = I;
            END;


                /* PROCESS PULL_UD WIRED-LOGIC.              */

            OTHERWISE DO;
%INCLUDE TGCJOP;
            END;


           END;
          END;

          CALL GENERATE_TEST(SET_VALUE);

          IF CRIT_VALUE = CRITICAL_ZERO
          THEN CRIT_VALUE = CRITICAL_ONE;
          ELSE CRIT_VALUE = CRITICAL_ZERO;

          IF PRE_INIT THEN DO;
            IF TEST_FLAG(QUEUE(ONE,ONE)) = ALL_TEST THEN DO;
              PRIV_OP->NEXT_OP = OP->NEXT_OP;
              OP = PRIV_OP->NEXT_OP;
              CRIT_VALUE = CRITICAL_ZERO;
              SET_VALUE = ONE;
            END;
            ELSE IF CRIT_VALUE = CRITICAL_ZERO | SET_VALUE = TWO THEN DO;
              PRIV_OP = OP;
              OP = OP->NEXT_OP;
              CRIT_VALUE = CRITICAL_ZERO;
              SET_VALUE = ONE;
            END;
            PRE_INIT = NO;
          END;

          ELSE IF PRIM_QTOP ¬= ZERO THEN DO;
            IF SET_VALUE = TWO THEN DO;
              IF TEST_FLAG(QUEUE(ONE,ONE)) = ALL_TEST THEN DO;
                PRIV_OP->NEXT_OP = OP->NEXT_OP;
                OP = PRIV_OP->NEXT_OP;
              END;
              ELSE DO;
                PRIV_OP = OP;
                OP = OP->NEXT_OP;
              END;
              CRIT_VALUE = CRITICAL_ZERO;
              SET_VALUE = ONE;
            END;
            ELSE SET_VALUE = TWO;
          END;

          ELSE IF TEST_FLAG(QUEUE(ONE,ONE)) = ALL_TEST THEN DO;
            PRIV_OP->NEXT_OP = OP->NEXT_OP;
          OP = PRIV_OP->NEXT_OP;
          CRIT_VALUE = CRITICAL_ZERO;
          SET_VALUE = ONE;
        END;

        ELSE IF CRIT_VALUE = CRITICAL_ZERO | SET_VALUE = TWO THEN DO;
          PRIV_OP = OP;
          OP = OP->NEXT_OP;
          CRIT_VALUE = CRITICAL_ZERO;
          SET_VALUE = ONE;
        END;


        ELSE IF SET_VALUE = ONE & CRIT_VALUE = CRITICAL_ONE
         & (TEST_FLAG(OP->LINEOUT_NO) & (CRIT1 | HIGHP)) =
         (CRIT1 | HIGHP) THEN DO;
          PRIV_OP = OP;
          OP = OP->NEXT_OP;
          CRIT_VALUE = CRITICAL_ZERO;
        END;

       END;

    END;

    RETURN(NO);

 END PROCESS_NETWORK;

 /**TGOUTPUT**/
 OUTPUT_DATA: PROC;

       /*************************************************************
        *                                                           *
        *   This routine prints the input wGS network in the output  *
        * trace file "FTRACE."                                      *
        *                                                           *
        *************************************************************/


    CALL TRACE('TGOUTPUT');

    PUT FILE(FTRACE) SKIP(2) EDIT('** INPUT NETWORK.')(COL(2),A);

    DO I = 1 TO MAX_ELEMENT;

       SELECT(TYPE(I));

             /* A POSITIVE SWITCH IN THE STRUCTURE.     */

          WHEN(POSITIVE_SWITCH) DO;
             PUT FILE(FTRACE) SKIP EDIT('PS',I,
              INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
          END;


             /* A NEGATIVE SWITCH IN THE STRUCTURE.      */

          WHEN(NEGATIVE_SWITCH) DO;
             PUT FILE(FTRACE) SKIP EDIT('NS',I,
              INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
          END;
```

```
   /* A GENERAL WIRED-LOGIC IN THE STRUCTURE.      */

WHEN(GENERAL_WIRED) DO;
   PUT FILE(FTRACE) SKIP EDIT('WG',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* A PULL-UP/DOWN WIRED-LOGIC IN THE STRUCTURE. */

WHEN(PULL_UD_WIRED) DO;
   PUT FILE(FTRACE) SKIP EDIT('WP',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* A HIGH WIRED-LOGIC IN THE STRUCTURE. */

WHEN(HIGH_WIRED) DO;
   PUT FILE(FTRACE) SKIP EDIT('WH',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* A LOW WIRED-LOGIC IN THE STRUCTURE. */

WHEN(LOW_WIRED) DO;
   PUT FILE(FTRACE) SKIP EDIT('WL',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* A COMPLIMENTARY WIRED-LOGIC IN THE STRUCTURE. */

WHEN(COMPLIMENTARY_WIRED) DO;
   PUT FILE(FTRACE) SKIP EDIT('WC',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* AN PRIMITIVE-INPUT ELEMENT IN THE STRUCTURE. */

WHEN(PRIMITIVE_INPUT) DO;
   PUT FILE(FTRACE) SKIP EDIT('PI',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* AN INPUT POSITIVE SWITCH IN THE STRUCTURE. */

WHEN(INPUT_POSITIVE) DO;
   PUT FILE(FTRACE) SKIP EDIT('IP',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;


   /* AN INPUT NEGATIVE SWITCH IN THE STRUCTURE. */

WHEN(INPUT_NEGATIVE) DO;
   PUT FILE(FTRACE) SKIP EDIT('IN',I,
    INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
END;
```

```
                /* A BIPOSITIVE SWITCH IN THE STRUCTURE. */

             WHEN(BIPOSITIVE_SWITCH) DO;
                PUT FILE(FTRACE) SKIP EDIT('BP',I,
                 INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
             END;


                /* A BINEGATIVE SWITCH IN THE STRUCTURE. */

             WHEN(BINEGATIVE_SWITCH) DO;
                PUT FILE(FTRACE) SKIP EDIT('BN',I,
                 INLINE(I),CNTLINE(I))(COL(2),A,F(8),F(8),F(8));
             END;


            OTHERWISE DO;
               PUT FILE(FTRACE) SKIP EDIT('UNIDENTIFIED ELEMENT',I)
               (COL(2),A,F(8));
            END;

         END;

      END;

      IP = INPUTS_HEAD->NEXT_IP;
      OP = OUTPUTS_HEAD->NEXT_OP;

      I = 1;
      DO WHILE(IP ¬= NULL);
         PUT FILE(FTRACE) SKIP EDIT('INPUT',I,IP->LINEIN_NO)
         (COL(2),A,F(8),F(8));
         I = I + 1;
         IP = IP->NEXT_IP;
      END;

      I = 1;
      DO WHILE(OP ¬= NULL);
         PUT FILE(FTRACE) SKIP EDIT('OUTPUT',I,OP->LINEOUT_NO)
         (COL(2),A,F(8),F(8));
         I = I + 1;
         OP = OP->NEXT_OP;
      END;

      PUT FILE(FTRACE) SKIP EDIT('** END OF INPUT NETWORK.')(COL(2),A);
      PUT FILE(FTRACE) SKIP;

   END OUTPUT_DATA;

   /**TGTRACE**/
   TRACE: PROC(STR);

         /***************************************************************
          *                                                             *
          *   This routine prints the trace message into the output     *
          * trace file "FTRACE."                                        *
          *                                                             *
          ***************************************************************/


      DCL  STR                   CHAR(*),
           OUT_LOGIC             CHAR(50) VAR;

      IF ELEMENT_NO = ZERO
   THEN OUT_LOGIC = 'DONT_CARE';
   ELSE
   SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

    WHEN(CRITICAL_ONE)
     OUT_LOGIC = 'CRITICAL_ONE';

    WHEN(CRITICAL_ZERO)
     OUT_LOGIC = 'CRITICAL_ZERO';

    WHEN(LOGIC_ZERO)
     OUT_LOGIC = 'LOGIC_ZERO';

    WHEN(LOGIC_ONE)
     OUT_LOGIC = 'LOGIC_ONE';

    WHEN(HIGHO_IMPEDANCE)
     OUT_LOGIC = 'HIGHO_IMPEDANCE';

    WHEN(HIGH1_IMPEDANCE)
     OUT_LOGIC = 'HIGH1_IMPEDANCE';

    WHEN(HIGH_IMPEDANCE)
     OUT_LOGIC = 'HIGH_IMPEDANCE';

    OTHERWISE
     OUT_LOGIC = 'DONT_CARE';
   END;

   PUT FILE(FTRACE) SKIP EDIT(STR,',','ELMT#',ELEMENT_NO,
    '  OUT=',OUT_LOGIC,'S=',STACK_TOP,'  Q=',QUEUE_TOP)(COL(2),A,
    A,COL(24),A,P'ZZZZZZ9',A,A,COL(60),A,P'ZZZZZZ9',A,P'ZZZZZZ9');

END TRACE;

/**TGWCCFP**/
CONFLECT_WC_PROCESS: PROC(VALUE);

      /*****************************************************************
       *                                                               *
       *   This routine sets the test-flag for the current element    *
       * and changes the output of the element to the logic value     *
       * defined as "VALUE" (i.e. if the logic value of the current   *
       * element is defferent from "VALUE")  while returning to the   *
       * previous element.                                             *
       *                                                               *
       *****************************************************************/


   DCL VALUE                BIT(*);

   STACK_TOP = STACK_TOP - 1;
   TEST_FLAG(ELEMENT_NO) =
      (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRITO)
      & (TEST_FLAG(INLINE(ELEMENT_NO)) | CRIT1);

   IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= VALUE
   THEN DO;
    PREDIR(STACK_TOP) = AHEAD;
    OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;
   END;

IF TRACE_FLAG THEN CALL TRACE('TGWCCFP');
```

```
END CONFLECT_WC_PROCESS;

/**TGWHCFP**/
CONFLECT_WH_PROCESS: PROC(VALUE);

      /****************************************************************
       *                                                              *
       *   This routine sets the test-flag for the current element    *
       * and changes the output of the element to the logic value     *
       * defined as "VALUE" (i.e. if the logic value of the current   *
       * element is defferent from "VALUE") while returning to the    *
       * previous element.                                            *
       *                                                              *
       ****************************************************************/


   DCL VALUE               BIT(*);

   STACK_TOP = STACK_TOP - 1;
   TEST_FLAG(ELEMENT_NO) = TEST_FLAG(INLINE(ELEMENT_NO))
               & (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRITO);

   IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= VALUE
   THEN DO;
    PREDIR(STACK_TOP) = AHEAD;
    OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;
   END;

IF TRACE_FLAG THEN CALL TRACE('TGWHCFP');

END CONFLECT_WH_PROCESS;

/**TGWLCFP**/
CONFLECT_WL_PROCESS: PROC(VALUE);

      /****************************************************************
       *                                                              *
       *   This routine sets the test-flag for the current element    *
       * and changes the output of the element to the logic value     *
       * defined as "VALUE" (i.e. if the logic value of the current   *
       * element is defferent from "VALUE") while returning to the    *
       * previous element.                                            *
       *                                                              *
       ****************************************************************/


   DCL VALUE               BIT(*);

   STACK_TOP = STACK_TOP - 1;
   TEST_FLAG(ELEMENT_NO) = TEST_FLAG(INLINE(ELEMENT_NO))
              & (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRIT1);

   IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= VALUE
   THEN DO;
    PREDIR(STACK_TOP) = AHEAD;
    OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;
   END;

IF TRACE_FLAG THEN CALL TRACE('TGWLCFP');

END CONFLECT_WL_PROCESS;

/**TGWPCFR**/
CONFLECT_WP_RETURN: PROC(VALUE);
```

```
        /************************************************************
         *                                                          *
         *   This routine sets the test-flag for the current element   *
         * and changes the output of the element to the logic value    *
         * defined as "VALUE" while returning to the previous element. *
         *                                                          *
         ************************************************************/


   DCL VALUE                BIT(*);

   STACK_TOP = STACK_TOP - 1;
   PREDIR(STACK_TOP) = AHEAD;
   OUT_VALUE(ELEMENT_NO,QSELECT) = VALUE;

   TEST_FLAG(ELEMENT_NO) =
               TEST_FLAG(ELEMENT_NO) | TEST_FLAG(NEXT_ELMT);

IF TRACE_FLAG THEN CALL TRACE('TGWPCFR');

END CONFLECT_WP_RETURN;

/**TGCJOP**/
   STACK_TOP = ONE;
   STACK(STACK_TOP) = ELMT_NO;
   IF JUSTIFY_OTHER_LINE THEN RETURN(YES);

/**TGCNCE**/
   STACK_TOP = STACK_TOP + 1;
   QUEUE_TOP = QUEUE_TOP + 1;
   STACK(STACK_TOP),QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
   PREDIR(STACK_TOP) = AHEAD;

/**TGCNE**/
   STACK_TOP = STACK_TOP + 1;
   STACK(STACK_TOP) = NEXT_ELMT;
   PREDIR(STACK_TOP) = AHEAD;

/**TGDCL**/
/* START       GLOBAL DECLARATION OF AN ELEMENT.
*/

           /*   E L E M E N T S   S T R U C T U R E   */

   DCL 1 ELEMENT(MAX_ELEMENT)              CONTROLLED,

         2 TYPE              BIT(4)         INIT('0000'B),
         2 OUT_VALUE(2)      BIT(4)
                             INIT( (MAX_ELEMENT*2) '0000'B),
         2 TEST_FLAG         BIT(4)         INIT('0000'B),
         2 CNTLINE           FIXED BIN(31)  INIT(0),
         2 INLINE            FIXED BIN(31)  INIT(0),

           /* STACK AND QUEUE FOR THE PROGRAM. */

       1 PATH_STACK(0:MAX_STACK)             CONTROLLED,
         2 STACK             FIXED BIN(31)  INIT(0),
         2 PREDIR            BIT(2)         INIT('00'B),
       STACK_TOP             FIXED BIN(31)  INIT(0),

       1 PATH_QUEUE(MAX_QUEUE)              CONTROLLED,
         2 QUEUE(2)          FIXED BIN(31)  INIT(0),
QUEUE_TOP            FIXED BIN(31)  INIT(0),
TEMP_QTOP            FIXED BIN(31)  INIT(0),
QSELECT              FIXED BIN(31)  INIT(1),


    /* SOME VARIABLE TO BE USED FOR MAX. STORAGE */

MAX_ELEMENT          FIXED BIN(31)  INIT(0),
MAX_STACK            FIXED BIN(31)  INIT(0),
MAX_QUEUE            FIXED BIN(31)  INIT(0),


    /* DEFINE STORAGE FOR PRIMITIVE INPUTS.     */

1 PRIMITIVE_INPUTS                  BASED(INPUTS_HEAD),
  2 LINEIN_NO        FIXED BIN(31)  INIT(0),
  2 NEXT_IP          POINTER,
INPUTS_HEAD          POINTER,
IP                   POINTER,       /* INPUT POINTER  */


    /* DEFINE STORAGE FOR PRIMITIVE OUTPUTS.    */

1 PRIMITIVE_OUTPUTS                 BASED(OUTPUTS_HEAD),
  2 LINEOUT_NO       FIXED BIN(31)  INIT(0),
  2 PATH_FLAG        BIT(1)         INIT('0'B),
  2 NEXT_OP          POINTER,
OUTPUTS_HEAD         POINTER,
OP                   POINTER,       /* OUTPUT POINTER */
PRIV_OP              POINTER,


    /* DEFINED VALUE FOR ELEMENT.TYPE           */

PRIMITIVE_INPUT      BIT(4)         INIT('0010'B),
POSITIVE_SWITCH      BIT(4)         INIT('0100'B),
NEGATIVE_SWITCH      BIT(4)         INIT('0110'B),
INPUT_POSITIVE       BIT(4)         INIT('1000'B),
INPUT_NEGATIVE       BIT(4)         INIT('1010'B),
BIPOSITIVE_SWITCH    BIT(4)         INIT('1100'B),
BINEGATIVE_SWITCH    BIT(4)         INIT('1110'B),
PULL_UD_WIRED        BIT(4)         INIT('0001'B),
COMPLIMENTARY_WIRED  BIT(4)         INIT('0011'B),
LOW_WIRED            BIT(4)         INIT('0101'B),
HIGH_WIRED           BIT(4)         INIT('0111'B),
GENERAL_WIRED        BIT(4)         INIT('1111'B),


    /* DEFINED VALUE FOR ELEMENT.OUT_VALUE      */

DONT_CARE            BIT(4)         INIT('0000'B),
HIGH_IMPEDANCE       BIT(4)         INIT('0001'B),
HIGHO_IMPEDANCE      BIT(4)         INIT('0010'B),
HIGH1_IMPEDANCE      BIT(4)         INIT('0011'B),
LOGIC_ZERO           BIT(4)         INIT('0100'B),
LOGIC_ONE            BIT(4)         INIT('0101'B),
CRITICAL_ZERO        BIT(4)         INIT('0110'B),
CRITICAL_ONE         BIT(4)         INIT('0111'B),


    /* DEFINED VALUE FOR ELEMENT.INLINE,CNTLINE */

GND                  FIXED BIN(31)  INIT(0),
VDD                  FIXED BIN(31)  INIT(1),
        WEAK_ZERO               FIXED BIN(31)  INIT(0),
        WEAK_ONE                FIXED BIN(31)  INIT(1),
        BISWITCH_CNT            FIXED BIN(31)  INIT(0),


            /* DEFINED VALUE FOR ELEMENT.TEST_FLAG      */

        HIGH_VALUE              BIT(4)         INIT('0000'B),
        CRIT0                   BIT(4)         INIT('0001'B),
        CRIT1                   BIT(4)         INIT('0010'B),
        HIGH                    BIT(4)         INIT('0100'B),
        HIGH_PATH               BIT(4)         INIT('1000'B),
        HIGHP                   BIT(4)         INIT('1100'B),

        NO_INIT                 BIT(4)         INIT('0000'B),
        WD_INIT                 BIT(4)         INIT('0001'B),
        WU_INIT                 BIT(4)         INIT('0010'B),
        ALL_INIT                BIT(4)         INIT('1111'B),
        NO_TEST                 BIT(4)         INIT('0000'B),
        ALL_TEST                BIT(4)         INIT('1111'B),


            /* DEFINED VALUE FOR STACK_DIR              */

        BACK                    BIT(2)         INIT('00'B),
        AHEAD                   BIT(2)         INIT('01'B),
        CNTDIR                  BIT(2)         INIT('10'B),
        INDIR                   BIT(2)         INIT('11'B),

/* END
*/

/* START           GENERAL CLOBAL DECLARATION.
*/

            /* GENERAL DEFINED VALUE FOR OTHER FLAGS    */

        EOF                     BIT(1)         INIT('0'B),
        NO                      BIT(1)         INIT('0'B),
        YES                     BIT(1)         INIT('1'B),


            /* BUILTIN FUNCTIONS DECLARATION.           */

        NULL                    BUILTIN,
        PLIRETC                 BUILTIN,
        SUBSTR                  BUILTIN,
        INDEX                   BUILTIN,


            /* FILES USED IN THE PROGRAM                */

        IN                      FILE           RECORD,
        OUT                     FILE           RECORD,
        FTRACE                  FILE,
        SYSPRINT                FILE,


            /* OTHER DECLARATION.                       */

        TRACE_FLAG              BIT(1)         INIT('0'B),
        COMMAND                 CHAR(2)        INIT('  '),
        DUMMY_CHAR              CHAR(1)        INIT(' '),
        TEXT                    CHAR(1920) VAR INIT(' '),
       TEST_NO              PIC 'ZZZZZZZ9' INIT(0),
       ZERO                 FIXED BIN(31)  INIT(0),
       ONE                  FIXED BIN(31)  INIT(1),
       TWO                  FIXED BIN(31)  INIT(2),
       ELEMENT_NO           FIXED BIN(31)  INIT(0),
       NEXT_ELMT            FIXED BIN(31)  INIT(0),
       OTHER_ELMT           FIXED BIN(31)  INIT(0),
       IN_NO                FIXED BIN(31)  INIT(0),
       CNT_NO               FIXED BIN(31)  INIT(0),
       NO_OUTPUTS           FIXED BIN(31)  INIT(0),
       NO_INPUTS            FIXED BIN(31)  INIT(0),
       (I,J,K)              FIXED BIN(31)  INIT(0);

/* END
*/

/**TGIN**/
/* START        PROCESS INPUT NEGATIVE SWITCH.
*/

     WHEN(INPUT_NEGATIVE) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (IN,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE   */
        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);
        PREDIR(STACK_TOP) = CNTDIR;
        SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

         WHEN(DONT_CARE) DO;
           /* WHEN CNTLINE IS DONT CARE */
          SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

           WHEN(HIGH0_IMPEDANCE,HIGH1_IMPEDANCE)
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
           WHEN(HIGH_IMPEDANCE)
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
           OTHERWISE
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;

          END;
%INCLUDE TGCNE;
 QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
         END;

         WHEN(LOGIC_ONE) DO;
           /* WHEN CNTLINE=LOGIC_ONE */
          SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

           WHEN(HIGH_IMPEDANCE)
            STACK_TOP = STACK_TOP - 1;

           WHEN(HIGH0_IMPEDANCE,HIGH1_IMPEDANCE) DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
           END;

           OTHERWISE
             CALL CONFLECT_IS_RETURN(HIGH_IMPEDANCE);

          END;
         END;
```

```
WHEN(CRITICAL_ONE) DO;
  /* WHEN CNTLINE=CRITICAL_ONE */
 SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

  WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
   STACK_TOP = STACK_TOP - 1;

  OTHERWISE DO;
   IF OTHER_ELMT = O THEN
    CALL CONFLECT_IS_RETURN(HIGHO_IMPEDANCE);
   ELSE
    CALL CONFLECT_IS_RETURN(HIGH1_IMPEDANCE);
  END;

 END;
 TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGHP;
END;

WHEN(LOGIC_ZERO) DO;
  /* WHEN CNTLINE=LOGIC_ZERO */
 SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

  WHEN(LOGIC_ONE,LOGIC_ZERO)
   STACK_TOP = STACK_TOP - 1;

  OTHERWISE DO;
   IF OTHER_ELMT = ZERO THEN
    CALL CONFLECT_IS_RETURN(LOGIC_ZERO);
   ELSE
    CALL CONFLECT_IS_RETURN(LOGIC_ONE);
  END;

 END;
END;

WHEN(CRITICAL_ZERO) DO;
  /* WHEN CNTLINE=CRITICAL_ZERO */
 SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

  WHEN(CRITICAL_ONE) DO;
   TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;
   STACK_TOP = STACK_TOP - 1;
  END;


  WHEN(CRITICAL_ZERO) DO;
   TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;
   STACK_TOP = STACK_TOP - 1;
  END;

  OTHERWISE DO;
   IF OTHER_ELMT = ZERO THEN DO;
    TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;
    CALL CONFLECT_IS_RETURN(CRITICAL_ZERO);
   END;
   ELSE DO;
    TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;
    CALL CONFLECT_IS_RETURN(CRITICAL_ONE);
   END;
  END;

 END;
END;
```

```
          OTHERWISE
            /* IMPROPER PRESET */
           RETURN(YES);

         END;
        END;

        OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (IN,RETURN)');
          /* PROCESS RETURN FROM THE ELEMENT */
         SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

          WHEN(CRITICAL_ZERO)
           TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;

          WHEN(CRITICAL_ONE)
           TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;

          WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
           IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ONE
            THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGHP;

          OTHERWISE;

         END;
         STACK_TOP = STACK_TOP - 1;
        END;
          .
       END;
      END;

/*END
*/

/**TGIP**/
/* START        PROCESS INPUT POSITIVE SWITCH.
*/

      WHEN(INPUT_POSITIVE) DO;
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (IP,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE   */
         NEXT_ELMT = CNTLINE(ELEMENT_NO);
         OTHER_ELMT = INLINE(ELEMENT_NO);
         PREDIR(STACK_TOP) = CNTDIR;
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
            /* WHEN CNTLINE IS DONT CARE */
           SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

            WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
            WHEN(HIGH_IMPEDANCE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
            OTHERWISE
                OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;

           END;
%INCLUDE TGCNE;
 QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
```

```
           END;

           WHEN(LOGIC_ZERO) DO;
             /* WHEN CNTLINE=LOGIC_ZERO */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGH_IMPEDANCE)
              STACK_TOP = STACK_TOP - 1;

             WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
             END;

             OTHERWISE
               CALL CONFLECT_IS_RETURN(HIGH_IMPEDANCE);

            END;
           END;

           WHEN(CRITICAL_ZERO) DO;
             /* WHEN CNTLINE=LOGIC,CRITICAL_ZERO */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
              STACK_TOP = STACK_TOP - 1;

             OTHERWISE DO;
              IF OTHER_ELMT = O THEN
               CALL CONFLECT_IS_RETURN(HIGHO_IMPEDANCE);
              ELSE
               CALL CONFLECT_IS_RETURN(HIGH1_IMPEDANCE);
             END;

            END;
            TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGHP;
           END;

           WHEN(LOGIC_ONE) DO;
             /* WHEN CNTLINE=LOGIC_ONE */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(LOGIC_ONE,LOGIC_ZERO)
              STACK_TOP = STACK_TOP - 1;

             OTHERWISE DO;
              IF OTHER_ELMT = ZERO THEN
               CALL CONFLECT_IS_RETURN(LOGIC_ZERO);
              ELSE
               CALL CONFLECT_IS_RETURN(LOGIC_ONE);
             END;

            END;
           END;

           WHEN(CRITICAL_ONE) DO;
             /* WHEN CNTLINE=CRITICAL_ONE */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(CRITICAL_ONE) DO;
              TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;
              STACK_TOP = STACK_TOP - 1;
             END;
             WHEN(CRITICAL_ZERO) DO;
              TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;
              STACK_TOP = STACK_TOP - 1;
             END;

             OTHERWISE DO;
              IF OTHER_ELMT = ZERO THEN DO;
               TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;
               CALL CONFLECT_IS_RETURN(CRITICAL_ZERO);
              END;
              ELSE DO;
               TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;
               CALL CONFLECT_IS_RETURN(CRITICAL_ONE);
              END;
             END;

            END;
           END;

           OTHERWISE
             /* IMPROPER PRESET */
            RETURN(YES);

          END;
         END;

         OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (IP,RETURN)');
           /* PROCESS RETURN FROM THE ELEMENT */
          SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

           WHEN(CRITICAL_ZERO)
            TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRITO;

           WHEN(CRITICAL_ONE)
            TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | CRIT1;

           WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
            IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ZERO
             THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGHP;

           OTHERWISE;

          END;
          STACK_TOP = STACK_TOP - 1;
         END;

        END;
       END;

/*END
*/

/**TGNS**/
/* START        PROCESS NEGATIVE SWITCH.
*/

        SELECT(PREDIR(STACK_TOP));

         WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (NS,AHEAD)');
           /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */
        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);

        SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

         WHEN(DONT_CARE) DO;
          PREDIR(STACK_TOP) = CNTDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
             /* WHEN BOTH INLINE AND CNTLINE ARE DONT CARE */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
              IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT
               | (TEST_FLAG(ELEMENT_NO) & HIGH) = NO_TEST
              THEN OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
              ELSE IF (TEST_FLAG(ELEMENT_NO) & HIGHP) = HIGHP THEN DO;
               PREDIR(STACK_TOP) = INDIR;
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
              END;
              ELSE OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
             END;

             WHEN(LOGIC_ZERO,LOGIC_ONE)
                 OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;

             WHEN(HIGH_IMPEDANCE) DO;
                 PREDIR(STACK_TOP) = INDIR;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
             END;

             OTHERWISE
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;

            END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;

           WHEN(LOGIC_ONE) DO;
             /* WHEN INLINE=DONT CARE & CNTLINE=LOGIC_ONE */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGH_IMPEDANCE) DO;
              STACK_TOP = STACK_TOP - 1;
              IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
              THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
               | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
             END;

             WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
              IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH THEN DO;
               STACK_TOP = STACK_TOP - 1;
               TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
               | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
              END;
              ELSE DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
              END;
             END;

             OTHERWISE
```

```
              CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);

            END;
           END;

           WHEN(CRITICAL_ONE) DO;
             /* WHEN INLINE=DONT CARE & CNTLINE=CRITICAL_ONE */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(CRITICAL_ONE,CRITICAL_ZERO,HIGH_IMPEDANCE) DO;
              PREDIR(STACK_TOP - 1) = AHEAD;
              OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
             END;

             WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
              IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH THEN DO;
               STACK_TOP = STACK_TOP - 1;
               TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
              END;
             END;

             WHEN(LOGIC_ONE,LOGIC_ZERO)
              CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);

             OTHERWISE;

            END;
           END;

           WHEN(LOGIC_ZERO) DO;
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
             WHEN(CRITICAL_ZERO,CRITICAL_ONE) DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
             END;
             WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
              IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT
              THEN RETURN(YES);
             OTHERWISE;
            END;
           END;

           WHEN(CRITICAL_ZERO) DO;
            IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT THEN
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
             WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
              RETURN(YES);
             OTHERWISE;
            END;
           END;

           OTHERWISE
             /* INVALID SETTING */
              RETURN(YES);

          END;
         END;

         WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
```

```
              /* WHEN INLINE=LOGIC,CRITICAL_ZERO & CNTLINE=DONT_CARE */
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

              WHEN(HIGH_IMPEDANCE) DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;

              WHEN(HIGHO_IMPEDANCE) DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;

              WHEN(LOGIC_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
               THEN CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ZERO);
               ELSE DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
               END;
              END;

              WHEN(CRITICAL_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN PREDIR(STACK_TOP) = CNTDIR;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;

              OTHERWISE  /* WHEN OUTLINE IS LOGIC_ONE, CRITICAL_ONE,
                   OR HIGH1_IMPEDANCE        */
               CALL CONFLECT_GO(LOGIC_ONE,HIGH_IMPEDANCE);

             END;
            END;

            WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
              /* WHEN INLINE=LOGIC,CRITICAL_ZERO &
                         & CNTLINE=LOGIC,CRITICAL_ONE */
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
              WHEN(HIGH_IMPEDANCE) DO;
               IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
               THEN CALL CONFLECT_RETURN(YES,HIGHO_IMPEDANCE);
               ELSE DO;
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                  | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;
              END;

              WHEN(HIGHO_IMPEDANCE) DO;
               IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
               THEN DO;
                TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                  | HIGH | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                STACK_TOP = STACK_TOP - 1;
               END;
               ELSE DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
```

```
%INCLUDE TGCNE;
               END;
              END;

              OTHERWISE DO;
               IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
               THEN CALL CONFLECT_RETURN(YES,HIGHO_IMPEDANCE);
               ELSE CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);
              END;

             END;
            END;

            WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
              /* WHEN INLINE=LOGIC,CRITICAL_ZERO &
                                          CNTLINE=LOGIC,CRITICAL_ZERO */
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

              WHEN(LOGIC_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN DO;
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;
               ELSE DO;
                IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
                THEN CALL CONFLECT_RETURN(YES,CRITICAL_ZERO);
                ELSE CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ZERO);
               END;
              END;

              WHEN(CRITICAL_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN PREDIR(STACK_TOP) = CNTDIR;
               IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO
               THEN DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
               END;
               ELSE IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
               THEN DO;
                TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                | (TEST_FLAG(OTHER_ELMT) & CRITO);
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;
              END;

              OTHERWISE DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN CALL CONFLECT_RETURN(NO,LOGIC_ZERO);
               ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
               THEN CALL CONFLECT_RETURN(YES,CRITICAL_ZERO);
               ELSE CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ZERO);
              END;

             END;
            END;

            OTHERWISE  /* CNTLINE IS ONE OF THE HIGH'S */
```

```
                /* INVALID SETTING */
               RETURN(YES);

              END;
             END;

             WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
              PREDIR(STACK_TOP) = INDIR;
              SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

               WHEN(DONT_CARE) DO;
                /* WHEN INLINE=LOGIC,CRITICAL_ONE & CNTLINE=DONT_CARE */
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                WHEN(HIGH_IMPEDANCE) DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                WHEN(HIGH1_IMPEDANCE) DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                WHEN(LOGIC_ONE) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
                 THEN DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                 END;
                 ELSE CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ONE);
                END;

                WHEN(CRITICAL_ONE) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
                 THEN PREDIR(STACK_TOP) = CNTDIR;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                OTHERWISE  /* WHEN OUTLINE IS LOGIC_ZERO,CRITICAL_ZERO,
                     OR HIGHO_IMPEDANCE        */
                 CALL CONFLECT_GO(LOGIC_ONE,HIGH_IMPEDANCE);

               END;
              END;

              WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
                /* WHEN INLINE=LOGIC,CRITICAL_ONE &
                                         CNTLINE=LOGIC,CRITICAL_ONE */
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                WHEN(HIGH_IMPEDANCE) DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
                 THEN CALL CONFLECT_RETURN(YES,HIGH1_IMPEDANCE);
                 ELSE DO;
                  STACK_TOP = STACK_TOP - 1;
                  IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                  THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                 END;
                END;

                WHEN(HIGH1_IMPEDANCE) DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
                 THEN DO;
                  TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | HIGH | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                  STACK_TOP = STACK_TOP - 1;
                 END;
                 ELSE DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
                 END;
                END;

                OTHERWISE DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
                 THEN CALL CONFLECT_RETURN(YES,HIGH1_IMPEDANCE);
                 ELSE CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);
                END;

               END;
              END;

              WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
                  /* WHEN BOTH INLINE & CNTLINE=LOGIC,CRITICAL_ONE */
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                WHEN(LOGIC_ONE) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
                 THEN DO;
                  STACK_TOP = STACK_TOP - 1;
                  IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                  THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                 END;
                 ELSE DO;
                  IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE
                  THEN CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ONE);
                  ELSE CALL CONFLECT_RETURN(YES,CRITICAL_ONE);
                 END;
                END;

                WHEN(CRITICAL_ONE) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
                 THEN PREDIR(STACK_TOP) = CNTDIR;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO
                 THEN DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
                 END;
                 ELSE IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ONE
                 THEN DO;
                  TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | (TEST_FLAG(OTHER_ELMT) & CRIT1);
                  STACK_TOP = STACK_TOP - 1;
                  IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                  THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                 END;
                END;

                OTHERWISE DO;
```

```
                   IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
                   THEN CALL CONFLECT_RETURN(NO,LOGIC_ONE);
                   ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
                   THEN CALL CONFLECT_RETURN(YES,CRITICAL_ONE);
                   ELSE CALL CONFLECT_GO(CRITICAL_ZERO,CRITICAL_ONE);
                  END;

                 END;
                END;

                OTHERWISE
                  /* INVALID SETTING */
                   RETURN(YES);

               END;
              END;

              OTHERWISE DO;
               /* WHEN INLINE IS ONE OF THE HIGH'S */
               PREDIR(STACK_TOP) = INDIR;
               SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

                WHEN(DONT_CARE) DO;
                 SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                  WHEN(HIGH_IMPEDANCE) DO;

                   OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
                  END;

                  WHEN(OUT_VALUE(OTHER_ELMT,QSELECT)) DO;

                   OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
                  END;

                  OTHERWISE DO; /* WHEN OUTLINE IS NOT HIGH_IMPEDANCE OR
                       SAME AS INLINE.           */
                   OUT_VALUE(ELEMENT_NO,QSELECT) =
                                          OUT_VALUE(OTHER_ELMT,QSELECT);
                   PREDIR(STACK_TOP - 1) = AHEAD;
                   OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
                  END;

                 END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
                  /* WHEN INLINE=ANY-HIGH'S & CNTLINE=LOGIC,CRITICAL_ONE */
                 SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                  WHEN(HIGH_IMPEDANCE) DO;
                   STACK_TOP = STACK_TOP - 1;
                   IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                   THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                    | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                  END;

                  OTHERWISE
                   CALL CONFLECT_RETURN(NO,HIGH_IMPEDANCE);

                 END;
                END;
```

```
            WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;

             IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                       OUT_VALUE(OTHER_ELMT,QSELECT)
             THEN DO;
              STACK_TOP = STACK_TOP - 1;
              IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
              THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
               | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
             END;

             ELSE
              CALL CONFLECT_RETURN(NO,OUT_VALUE(OTHER_ELMT,QSELECT));

            END;

            OTHERWISE
              /* INVALID SETTING  */
             RETURN(YES);

           END;
          END;

         END;
        END;

        WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (NS,CNTDIR)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */

         PREDIR(STACK_TOP) = INDIR;
         NEXT_ELMT = INLINE(ELEMENT_NO);
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           SELECT(OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT));

            WHEN(CRITICAL_ONE) DO;
             IF OUT_VALUE(ELEMENT_NO,QSELECT) = HIGHO_IMPEDANCE
             THEN
              OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
             ELSE
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
            END;

            OTHERWISE
             OUT_VALUE(NEXT_ELMT,QSELECT) =
                                         OUT_VALUE(ELEMENT_NO,QSELECT);

           END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
          END;

          WHEN(LOGIC_ONE) DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
          END;

          OTHERWISE DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
          END;
         END;
        END;

        OTHERWISE DO:
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (NS,RETURN)');
          /* PROCESS RETURN FROM THIS ELEMENT */
         SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

          WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
           IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ONE
            THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGH;

          WHEN(CRITICAL_ZERO)
            TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
             | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRITO);

          WHEN(CRITICAL_ONE)
            TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
             | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRIT1);

          OTHERWISE;

         END;

         IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
          THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
           | (TEST_FLAG(INLINE(ELEMENT_NO)) & HIGH_PATH);
         STACK_TOP = STACK_TOP - 1;
        END;

       END;

/*END
*/

/**TGPS**/
/* START        PROCESS POSITIVE SWITCH.
*/

       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (PS,AHEAD)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */

         NEXT_ELMT = CNTLINE(ELEMENT_NO);
         OTHER_ELMT = INLINE(ELEMENT_NO);

         SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           PREDIR(STACK_TOP) = CNTDIR;
           SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

            WHEN(DONT_CARE) DO;
              /* WHEN BOTH INLINE AND CNTLINE ARE DONT CARE */
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

              WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
               IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT
                | (TEST_FLAG(ELEMENT_NO) & HIGH) = NO_TEST
               THEN OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
               ELSE IF (TEST_FLAG(ELEMENT_NO) & HIGHP) = HIGHP THEN DO;
```

```
                 PREDIR(STACK_TOP) = INDIR;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
                END;
                ELSE OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
               END;

               WHEN(LOGIC_ZERO,LOGIC_ONE)
                   OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;

               WHEN(HIGH_IMPEDANCE) DO;
                   PREDIR(STACK_TOP) = INDIR;
                   OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
               END;

               OTHERWISE
                   OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;

              END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;

             WHEN(LOGIC_ZERO) DO;
               /* WHEN INLINE=DONT CARE & CNTLINE=LOGIC_ZERO */
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

               WHEN(HIGH_IMPEDANCE) DO;
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;

               WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH THEN DO;
                 STACK_TOP = STACK_TOP - 1;
                 TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                  | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                END;
                ELSE DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
                END;
               END;

               OTHERWISE
                CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);

              END;
             END;

             WHEN(CRITICAL_ZERO) DO;
               /* WHEN INLINE=DONT CARE & CNTLINE=CRITICAL_ZERO */
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

               WHEN(CRITICAL_ONE,CRITICAL_ZERO,HIGH_IMPEDANCE) DO;
                PREDIR(STACK_TOP - 1) = AHEAD;
                OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
               END;

               WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE) DO;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH THEN DO;
                 STACK_TOP = STACK_TOP - 1;
                 TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                END;
               END;

               WHEN(LOGIC_ONE,LOGIC_ZERO)
                CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);

               OTHERWISE;

              END;
             END;

             WHEN(LOGIC_ONE) DO;
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
               WHEN(CRITICAL_ZERO,CRITICAL_ONE) DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
               END;
               WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
                IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT
                THEN RETURN(YES);
               OTHERWISE;
              END;
             END;

             WHEN(CRITICAL_ONE) DO;
              IF TYPE(OTHER_ELMT) = PRIMITIVE_INPUT THEN
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
               WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
                RETURN(YES);
               OTHERWISE;
              END;
             END;

             OTHERWISE
               /* INVALID SETTING */
                RETURN(YES);

            END;
           END;

           WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
            PREDIR(STACK_TOP) = INDIR;
            SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

             WHEN(DONT_CARE) DO;
               /* WHEN INLINE=LOGIC,CRITICAL_ZERO & CNTLINE=DONT_CARE */
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

               WHEN(HIGH_IMPEDANCE) DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
               END;

               WHEN(HIGHO_IMPEDANCE) DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
               END;

               WHEN(LOGIC_ZERO) DO;
                IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
                THEN CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ZERO);
```

```
                 ELSE DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                 END;
                END;

                WHEN(CRITICAL_ZERO) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
                 THEN PREDIR(STACK_TOP) = CNTDIR;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                OTHERWISE  /* WHEN OUTLINE IS LOGIC_ONE, CRITICAL_ONE,
                     OR HIGH1_IMPEDANCE        */
                 CALL CONFLECT_GO(LOGIC_ZERO,HIGH_IMPEDANCE);

               END;
              END;

              WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
                /* WHEN BOTH INLINE AND CNTLINE ARE LOGIC,CRITICAL_ZERO */
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
                WHEN(HIGH_IMPEDANCE) DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
                 THEN CALL CONFLECT_RETURN(YES,HIGHO_IMPEDANCE);
                 ELSE DO;
                  STACK_TOP = STACK_TOP - 1;
                  IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                  THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                 END;
                END;

                WHEN(HIGHO_IMPEDANCE) DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
                 THEN DO;
                  TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                   | HIGH | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
                  STACK_TOP = STACK_TOP - 1;
                 END;
                 ELSE DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
                 END;
                END;

                OTHERWISE DO;
                 IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
                 THEN CALL CONFLECT_RETURN(YES,HIGHO_IMPEDANCE);
                 ELSE CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);
                END;

               END;
              END;

              WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
                /* WHEN INLINE=LOGIC,CRITICAL_ZERO &
                                             CNTLINE=LOGIC,CRITICAL_ONE */
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                WHEN(LOGIC_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN DO;
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;
               ELSE DO;
                IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
                THEN CALL CONFLECT_RETURN(YES,CRITICAL_ZERO);
                ELSE CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ZERO);
               END;
              END;

              WHEN(CRITICAL_ZERO) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN PREDIR(STACK_TOP) = CNTDIR;
               IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE
               THEN DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
               END;
               ELSE IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
               THEN DO;
                TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                | (TEST_FLAG(OTHER_ELMT) & CRITO);
                STACK_TOP = STACK_TOP - 1;
                IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
                THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               END;
              END;

              OTHERWISE DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
               THEN CALL CONFLECT_RETURN(NO,LOGIC_ZERO);
               ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
               THEN  CALL CONFLECT_RETURN(YES,CRITICAL_ZERO);
               ELSE CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ZERO);
              END;

             END;
            END;

            OTHERWISE  /* CNTLINE IS ONE OF THE HIGH'S */
              /* INVALID SETTING */
             RETURN(YES);

           END;
          END;

          WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
           PREDIR(STACK_TOP) = INDIR;
           SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

            WHEN(DONT_CARE) DO;
              /* WHEN INLINE=LOGIC,CRITICAL_ONE CNTLINE=DONT_CARE */
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

              WHEN(HIGH_IMPEDANCE) DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;
             WHEN(HIGH1_IMPEDANCE) DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;

             WHEN(LOGIC_ONE) DO;
              IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
              THEN DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;
              ELSE CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ONE);
             END;

             WHEN(CRITICAL_ONE) DO;
              IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
              THEN PREDIR(STACK_TOP) = CNTDIR;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;

             OTHERWISE  /* WHEN OUTLINE IS LOGIC_ZERO,CRITICAL_ZERO,
                  OR HIGHO_IMPEDANCE         */
              CALL CONFLECT_GO(LOGIC_ZERO,HIGH_IMPEDANCE);

            END;
           END;

           WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
             /* WHEN INLINE=LOGIC,CRITICAL_ONE &
                                  CNTLINE=LOGIC,CRITICAL_ZERO */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGH_IMPEDANCE) DO;
              IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
              THEN CALL CONFLECT_RETURN(YES,HIGH1_IMPEDANCE);
              ELSE DO;
               STACK_TOP = STACK_TOP - 1;
               IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
               THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
              END;
             END;

             WHEN(HIGH1_IMPEDANCE) DO;
              IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
              THEN DO;
               TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
                 | HIGH | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
               STACK_TOP = STACK_TOP - 1;
              END;
              ELSE DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
              END;
             END;

             OTHERWISE DO;
              IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
              THEN CALL CONFLECT_RETURN(YES,HIGH1_IMPEDANCE);
```

```
          ELSE CALL CONFLECT_RETURN(YES,HIGH_IMPEDANCE);
         END;

        END;
       END;

       WHEN(LOGIC_ONE,CRITICAL_ONE) DO;
           /* WHEN BOTH INLINE & CNTLINE=LOGIC,CRITICAL_ONE */
        SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

         WHEN(LOGIC_ONE) DO;
          IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
          THEN DO;
           STACK_TOP = STACK_TOP - 1;
           IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
           THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
            | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
          END;
          ELSE DO;
           IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE
           THEN CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ONE);
           ELSE CALL CONFLECT_RETURN(YES,CRITICAL_ONE);
          END;
         END;

         WHEN(CRITICAL_ONE) DO;
          IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
          THEN PREDIR(STACK_TOP) = CNTDIR;
          IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE
          THEN DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
          END;
          ELSE IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ONE
          THEN DO;
           TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
            | (TEST_FLAG(OTHER_ELMT) & CRIT1);
           STACK_TOP = STACK_TOP - 1;
           IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
           THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
            | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
          END;
         END;

         OTHERWISE DO;
          IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
          THEN CALL CONFLECT_RETURN(NO,LOGIC_ONE);
          ELSE IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
          THEN CALL CONFLECT_RETURN(YES,CRITICAL_ONE);
          ELSE CALL CONFLECT_GO(CRITICAL_ONE,CRITICAL_ONE);
         END;

        END;
       END;

       OTHERWISE
         /* INVALID SETTING */
          RETURN(YES);

      END;
     END;

     OTHERWISE DO;
      /* WHEN INLINE IS ONE OF THE HIGH'S */
```

```
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGH_IMPEDANCE) DO;

              OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
             END;

             WHEN(OUT_VALUE(OTHER_ELMT,QSELECT)) DO;

              OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
             END;

             OTHERWISE DO; /* WHEN OUTLINE IS NOT HIGH_IMPEDANCE OR
                  SAME AS INLINE.           */
              OUT_VALUE(ELEMENT_NO,QSELECT) =
                                         OUT_VALUE(OTHER_ELMT,QSELECT);
              PREDIR(STACK_TOP - 1) = AHEAD;
              OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
             END;

            END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;

           WHEN(LOGIC_ZERO,CRITICAL_ZERO) DO;
             /* WHEN INLINE=ANY-HIGH'S & CNTLINE=LOGIC,CRITICAL_ZERO */
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(HIGH_IMPEDANCE) DO;
              STACK_TOP = STACK_TOP - 1;
              IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
              THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
               | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
             END;

             OTHERWISE
              CALL CONFLECT_RETURN(NO,HIGH_IMPEDANCE);

            END;
           END;

           WHEN(LOGIC_ONE,CRITICAL_ONE) DO;

            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                        OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             STACK_TOP = STACK_TOP - 1;
             IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
             THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
              | (TEST_FLAG(OTHER_ELMT) & HIGH_PATH);
            END;

            ELSE
             CALL CONFLECT_RETURN(NO,OUT_VALUE(OTHER_ELMT,QSELECT));

           END;

           OTHERWISE
             /* INVALID SETTING  */
```

```
              RETURN(YES);

            END;
           END;

          END;
         END;

         WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (PS,CNTDIR)');
           /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE */

          PREDIR(STACK_TOP) = INDIR;
          NEXT_ELMT = INLINE(ELEMENT_NO);
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            SELECT(OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT));

             WHEN(CRITICAL_ZERO) DO;
              IF OUT_VALUE(ELEMENT_NO,QSELECT) = HIGHO_IMPEDANCE
              THEN
               OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO;
              ELSE
                OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE;
             END;

             OTHERWISE
              OUT_VALUE(NEXT_ELMT,QSELECT) =
                                           OUT_VALUE(ELEMENT_NO,QSELECT);

            END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;

           WHEN(LOGIC_ONE) DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
           END;

           OTHERWISE DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
           END;

          END;
         END;

         OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (PS,RETURN)');
           /* PROCESS RETURN FROM THIS ELEMENT */
          SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

           WHEN(HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
            IF OUT_VALUE(CNTLINE(ELEMENT_NO),QSELECT) = CRITICAL_ZERO
             THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO) | HIGH;

           WHEN(CRITICAL_ZERO)
             TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
              | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRITO);

           WHEN(CRITICAL_ONE)
             TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
          | (TEST_FLAG(INLINE(ELEMENT_NO)) & CRIT1);

         OTHERWISE;

        END;

        IF (TEST_FLAG(ELEMENT_NO) & HIGH) = HIGH
         THEN TEST_FLAG(ELEMENT_NO) = TEST_FLAG(ELEMENT_NO)
          | (TEST_FLAG(INLINE(ELEMENT_NO)) & HIGH_PATH);
        STACK_TOP = STACK_TOP - 1;
       END;

      END;

/*END
*/

/**TGWC**/
/* START        PROCESS COMPLIMENTARY WIRED.
*/

     WHEN(COMPLIMENTARY_WIRED) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WC,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */

        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
        THEN HIGH_VALUE = HIGHO_IMPEDANCE;
        ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
        THEN HIGH_VALUE = HIGH1_IMPEDANCE;

        SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

         WHEN(CRITICAL_ZERO) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH1_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH1_IMPEDANCE,CRITICAL_ZERO);
           END;

           WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE)
            CALL CONFLECT_WC_PROCESS(CRITICAL_ZERO);

           OTHERWISE RETURN(YES);

          END;
         END;

         WHEN(LOGIC_ZERO) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));
```

```
               WHEN(DONT_CARE) DO;
                IF OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ZERO
                THEN DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;
                ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
                THEN DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
                 PREDIR(STACK_TOP) = CNTDIR;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;
                ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,LOGIC_ZERO);
               END;

               WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE)
                CALL CONFLECT_WC_PROCESS(LOGIC_ZERO);

               OTHERWISE RETURN(YES);

              END;
             END;

             WHEN(DONT_CARE) DO;
              SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

               WHEN(DONT_CARE) DO;
                SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                 WHEN(HIGH_IMPEDANCE) DO;
                  IF HIGH_VALUE = HIGHO_IMPEDANCE THEN
                                 NEXT_ELMT = OTHER_ELMT;
                  ELSE PREDIR(STACK_TOP) = CNTDIR;
                  OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
                  PREDIR(STACK_TOP - 1) = AHEAD;
                 END;

                 WHEN(CRITICAL_ZERO,LOGIC_ZERO,HIGHO_IMPEDANCE)
                  NEXT_ELMT = OTHER_ELMT;

                 OTHERWISE
                  PREDIR(STACK_TOP) = CNTDIR;

                END;
                  OUT_VALUE(NEXT_ELMT,QSELECT) =
                                             OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
               END;

               WHEN(HIGH1_IMPEDANCE) DO;
                IF OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE
                THEN DO;
                 OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
                 PREDIR(STACK_TOP - 1) = AHEAD;
                END;
                PREDIR(STACK_TOP) = CNTDIR;
               END;

               WHEN(HIGH_IMPEDANCE) DO;
                IF OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_IMPEDANCE
                 & HIGH_VALUE = HIGHO_IMPEDANCE
              THEN DO;
               OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
               PREDIR(STACK_TOP - 1) = AHEAD;
              END;
              PREDIR(STACK_TOP) = CNTDIR;
             END;

             WHEN(LOGIC_ONE) DO;
              PREDIR(STACK_TOP) = CNTDIR;
              IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
              THEN DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
              END;
              ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= LOGIC_ONE
              THEN DO;
               OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ONE;
               PREDIR(STACK_TOP - 1) = AHEAD;
              END;
             END;

             WHEN(CRITICAL_ONE) DO;
              PREDIR(STACK_TOP) = CNTDIR;
              IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= CRITICAL_ONE
              THEN DO;
               OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE;
               PREDIR(STACK_TOP - 1) = AHEAD;
              END;
             END;

             OTHERWISE RETURN(YES);

            END;
           END;

           WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE) DO;
            PREDIR(STACK_TOP) = INDIR;
            SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

             WHEN(DONT_CARE) DO;
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
               WHEN(HIGH_IMPEDANCE) DO;
                IF HIGH_VALUE = HIGH1_IMPEDANCE
                THEN CALL CONFLECT_GO(HIGH1_IMPEDANCE,HIGH1_IMPEDANCE);
                ELSE DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;
               END;
               WHEN(HIGHO_IMPEDANCE) DO;
                IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH_IMPEDANCE
                THEN CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH_IMPEDANCE);
                ELSE DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;
               END;
               WHEN(CRITICAL_ONE,LOGIC_ONE,HIGH1_IMPEDANCE) DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) =
                                        OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
```

```
                 END;
                 OTHERWISE DO:
                  IF HIGH_VALUE = HIGH1_IMPEDANCE
                  THEN CALL CONFLECT_GO(HIGH1_IMPEDANCE,HIGH1_IMPEDANCE);
                  ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                          OUT_VALUE(OTHER_ELMT,QSELECT));
                 END;

                END;
               END;

               WHEN(CRITICAL_ONE)
                CALL CONFLECT_WC_PROCESS(CRITICAL_ONE);

               WHEN(LOGIC_ONE) DO;
                IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
                THEN DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
                END;
                ELSE CALL CONFLECT_WC_PROCESS(LOGIC_ONE);
               END;

               WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE) DO;
                IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH_VALUE
                 | OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_VALUE
                THEN CALL CONFLECT_WC_PROCESS(HIGH_VALUE);
                ELSE CALL CONFLECT_WC_PROCESS(HIGH_IMPEDANCE);
               END;

               OTHERWISE RETURN(YES);

              END;
             END;

             OTHERWISE RETURN(YES);

            END;
           END;

           WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WC,CNTDIR)');
             /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */
            PREDIR(STACK_TOP) = INDIR;
            NEXT_ELMT = INLINE(ELEMENT_NO);
            OTHER_ELMT = CNTLINE(ELEMENT_NO);
            SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

             WHEN(DONT_CARE) DO;
              SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

               WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE)
                OUT_VALUE(NEXT_ELMT,QSELECT) =
                                          OUT_VALUE(ELEMENT_NO,QSELECT);

               WHEN(CRITICAL_ONE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH0_IMPEDANCE;

               WHEN(LOGIC_ONE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;

               OTHERWISE RETURN(YES);

              END;
```

```
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
         END;

         WHEN(LOGIC_ONE) DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
         END;

         OTHERWISE DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
         END;

        END;
       END;

       OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WC,RETURN)');
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;
        TEST_FLAG(ELEMENT_NO) =

          (TEST_FLAG(INLINE(ELEMENT_NO)) | CRIT1)
          & (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRIT0);
       END;

      END;
     END;

/*END
*/

/**TGWG**/
/* START        PROCESS GENERAL WIRED.
*/

     WHEN(GENERAL_WIRED) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WG,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */

        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
        THEN HIGH_VALUE = HIGH0_IMPEDANCE;
        ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
        THEN HIGH_VALUE = HIGH1_IMPEDANCE;

        SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

         WHEN(CRITICAL_ZERO,CRITICAL_ONE) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                      OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
```

```
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                           OUT_VALUE(OTHER_ELMT,QSELECT));
           END;

           WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
            CALL CONFLECT_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));

           WHEN(LOGIC_ONE) DO;
            IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
            THEN RETURN(YES);
            ELSE CALL CONFLECT_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
           END;

           WHEN(LOGIC_ZERO) DO;
            IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ONE
            THEN RETURN(YES);
            ELSE CALL CONFLECT_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
           END;

           OTHERWISE RETURN(YES);

          END;
         END;

         WHEN(LOGIC_ZERO,LOGIC_ONE) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                             OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE DO;
             IF (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
              &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE)
              | (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
              &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO)
             THEN DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
              PREDIR(STACK_TOP) = CNTDIR;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;
             ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                         OUT_VALUE(OTHER_ELMT,QSELECT));
            END;
           END;

           WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
            CALL CONFLECT_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));

           WHEN(CRITICAL_ONE) DO;
            IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
            THEN RETURN(YES);
            ELSE CALL CONFLECT_PROCESS(OUT_VALUE(NEXT_ELMT,QSELECT));
           END;
           WHEN(CRITICAL_ZERO) DO;
            IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
            THEN RETURN(YES);
            ELSE CALL CONFLECT_PROCESS(OUT_VALUE(NEXT_ELMT,QSELECT));
           END;

           OTHERWISE RETURN(YES);

          END;
         END;

         WHEN(DONT_CARE) DO;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

             WHEN(CRITICAL_ZERO) DO;
              IF (TEST_FLAG(NEXT_ELMT) & CRITO) = CRITO
               & (TEST_FLAG(OTHER_ELMT) & CRITO) = NO_TEST
              THEN NEXT_ELMT = OTHER_ELMT;
              ELSE PREDIR(STACK_TOP) = CNTDIR;
             END;

             WHEN(CRITICAL_ONE,LOGIC_ONE) DO;
              IF (TEST_FLAG(NEXT_ELMT) & CRIT1) = CRIT1
               & (TEST_FLAG(OTHER_ELMT) & CRIT1) = NO_TEST
              THEN NEXT_ELMT = OTHER_ELMT;
              ELSE PREDIR(STACK_TOP) = CNTDIR;
             END;

             OTHERWISE PREDIR(STACK_TOP) = CNTDIR;

            END;
            OUT_VALUE(NEXT_ELMT,QSELECT) =
                                         OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;

           WHEN(OUT_VALUE(ELEMENT_NO,QSELECT),HIGH_IMPEDANCE,
                HIGHO_IMPEDANCE,HIGH1_IMPEDANCE)
            PREDIR(STACK_TOP) = CNTDIR;

           WHEN(LOGIC_ONE) DO;
            PREDIR(STACK_TOP) = CNTDIR;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
            END;
            ELSE DO;
             OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ONE;
             PREDIR(STACK_TOP - 1) = AHEAD;
            END;
           END;

           WHEN(LOGIC_ZERO) DO;
            PREDIR(STACK_TOP) = CNTDIR;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
```

```
             END;
             ELSE DO;
              OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ZERO;
              PREDIR(STACK_TOP - 1) = AHEAD;
             END;
            END;

            OTHERWISE DO;
             PREDIR(STACK_TOP) = CNTDIR;
             OUT_VALUE(ELEMENT_NO,QSELECT) =
                                        OUT_VALUE(NEXT_ELMT,QSELECT);
             PREDIR(STACK_TOP - 1) = AHEAD;
            END;

           END;
          END;

          OTHERWISE DO;
            /* WHEN INLINE IS ONE OF THE HIGH'S */
           PREDIR(STACK_TOP) = INDIR;
           SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

            WHEN(DONT_CARE) DO;
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));
              WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),CRITICAL_ONE,
                   CRITICAL_ZERO,LOGIC_ONE,LOGIC_ZERO) DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) =
                                           OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;

              OTHERWISE
               CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH_IMPEDANCE);
             END;
            END;

            WHEN(CRITICAL_ZERO,CRITICAL_ONE,
                 OUT_VALUE(OTHER_ELMT,QSELECT))
             CALL CONFLECT_PROCESS(OUT_VALUE(NEXT_ELMT,QSELECT));

            WHEN(LOGIC_ONE) DO;
             IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
             THEN DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
             END;
             ELSE CALL CONFLECT_PROCESS(LOGIC_ONE);
            END;

            WHEN(LOGIC_ZERO) DO;
             IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
             THEN DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
             END;
             ELSE CALL CONFLECT_PROCESS(LOGIC_ZERO);
            END;

            OTHERWISE
             CALL CONFLECT_PROCESS(HIGH_IMPEDANCE);

           END;
          END;
          END;
         END;

         WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WG,CNTDIR)');
           /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE   */
          PREDIR(STACK_TOP) = INDIR;
          NEXT_ELMT = INLINE(ELEMENT_NO);
          OTHER_ELMT = CNTLINE(ELEMENT_NO);
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

             WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE,HIGHO_IMPEDANCE)
              OUT_VALUE(NEXT_ELMT,QSELECT) =
                                           OUT_VALUE(ELEMENT_NO,QSELECT);

             WHEN(OUT_VALUE(ELEMENT_NO,QSELECT))
              OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;

             OTHERWISE RETURN(YES);

            END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;

           WHEN(LOGIC_ONE) DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
           END;

           OTHERWISE DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
           END;

          END;
         END;

         OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WG,RETURN)');
           /* PROCESS RETURN FROM THIS ELEMENT */
          STACK_TOP = STACK_TOP - 1;
          TEST_FLAG(ELEMENT_NO) = TEST_FLAG(INLINE(ELEMENT_NO))
                                & TEST_FLAG(CNTLINE(ELEMENT_NO));
         END;

        END;
       END;

/*END
*/

/**TGWH**/
/* START        PROCESS HIGH WIRED.
*/

       WHEN(HIGH_WIRED) DO;
        SELECT(PREDIR(STACK_TOP));

         WHEN(AHEAD) DO;
```

```
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WH,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */

        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
        THEN HIGH_VALUE = HIGHO_IMPEDANCE;
        ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
        THEN HIGH_VALUE = HIGH1_IMPEDANCE;

        SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

         WHEN(CRITICAL_ZERO,CRITICAL_ONE) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                        OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                       OUT_VALUE(OTHER_ELMT,QSELECT));
           END;

           WHEN(LOGIC_ZERO,CRITICAL_ZERO)
            RETURN(YES);

           WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                                             HIGH1_IMPEDANCE) DO;
            IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO
            THEN RETURN(YES);
            ELSE
             CALL CONFLECT_WH_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
           END;

           WHEN(LOGIC_ONE) DO;
            IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ZERO
            THEN RETURN(YES);
            ELSE CALL CONFLECT_WH_PROCESS(CRITICAL_ONE);
           END;

           OTHERWISE RETURN(YES);

          END;
         END;

         WHEN(LOGIC_ZERO,LOGIC_ONE) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                        OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE DO;
            IF (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
             &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE)
             | (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
             &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
             PREDIR(STACK_TOP) = CNTDIR;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                    OUT_VALUE(OTHER_ELMT,QSELECT));
           END;
          END;

          WHEN(LOGIC_ZERO,CRITICAL_ZERO)
           RETURN(YES);

          WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                                             HIGH1_IMPEDANCE) DO;
           IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ZERO
           THEN RETURN(YES);
           ELSE
            CALL CONFLECT_WH_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
          END;

          WHEN(CRITICAL_ONE) DO;
           IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO
           THEN RETURN(YES);
           ELSE CALL CONFLECT_WH_PROCESS(CRITICAL_ONE);
          END;

          OTHERWISE RETURN(YES);

         END;
        END;

        WHEN(DONT_CARE) DO;
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

            WHEN(HIGH_IMPEDANCE) DO;
             IF HIGH_VALUE = HIGHO_IMPEDANCE THEN DO;
              NEXT_ELMT = OTHER_ELMT;
              OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
              PREDIR(STACK_TOP - 1) = AHEAD;
             END;
             ELSE PREDIR(STACK_TOP) = CNTDIR;
            END;

            WHEN(CRITICAL_ONE) DO;
             IF (TEST_FLAG(NEXT_ELMT) & CRIT1) = CRIT1
              & (TEST_FLAG(OTHER_ELMT) & CRIT1) = NO_TEST
             THEN NEXT_ELMT = OTHER_ELMT;
             ELSE PREDIR(STACK_TOP) = CNTDIR;
            END;

            WHEN(CRITICAL_ZERO,LOGIC_ZERO,HIGHO_IMPEDANCE)
             NEXT_ELMT = OTHER_ELMT;

            OTHERWISE PREDIR(STACK_TOP) = CNTDIR;
```

```
             END;
             OUT_VALUE(NEXT_ELMT,QSELECT) =
                                     OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;

            WHEN(CRITICAL_ONE,HIGH_IMPEDANCE,HIGH1_IMPEDANCE) DO;
             PREDIR(STACK_TOP) = CNTDIR;
             IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= CRITICAL_ONE
             THEN DO;
              OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE;
              PREDIR(STACK_TOP - 1) = AHEAD;
             END;
            END;

            WHEN(LOGIC_ONE) DO;
             PREDIR(STACK_TOP) = CNTDIR;
             IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
             THEN DO;
              OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
             END;
             ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= LOGIC_ONE
             THEN DO;
              OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ONE;
              PREDIR(STACK_TOP - 1) = AHEAD;
             END;
            END;

            OTHERWISE RETURN(YES);

           END;
          END;

          OTHERWISE DO;
            /* WHEN INLINE IS ONE OF THE HIGH'S */
           PREDIR(STACK_TOP) = INDIR;
           SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

            WHEN(DONT_CARE) DO;
             SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

              WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),CRITICAL_ONE,
                                                   LOGIC_ONE) DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) =
                                         OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
              END;

              WHEN(HIGH_IMPEDANCE) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGHO_IMPEDANCE
               THEN CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGHO_IMPEDANCE);
               ELSE DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
               END;
              END;

              OTHERWISE DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGHO_IMPEDANCE
               THEN CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGHO_IMPEDANCE);
```

```
                  ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH_IMPEDANCE);
                 END;

                END;
               END;

               WHEN(CRITICAL_ONE)
                CALL CONFLECT_WH_PROCESS(CRITICAL_ONE);

               WHEN(LOGIC_ONE) DO;
                IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
                THEN DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
                END;
                ELSE CALL CONFLECT_WH_PROCESS(LOGIC_ONE);
               END;

               WHEN(HIGH_IMPEDANCE) DO;
                IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGHO_IMPEDANCE
                THEN CALL CONFLECT_WH_PROCESS(HIGHO_IMPEDANCE);
                ELSE CALL CONFLECT_WH_PROCESS(HIGH_IMPEDANCE);
               END;

               WHEN(HIGH1_IMPEDANCE) DO;
                IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH_VALUE
                THEN CALL CONFLECT_WH_PROCESS(HIGH_VALUE);
                ELSE CALL CONFLECT_WH_PROCESS(HIGH_IMPEDANCE);
               END;

               OTHERWISE RETURN(YES);

              END;
             END;

            END;
           END;

           WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WH,CNTDIR)');
             /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */
            PREDIR(STACK_TOP) = INDIR;
            NEXT_ELMT = INLINE(ELEMENT_NO);
            OTHER_ELMT = CNTLINE(ELEMENT_NO);
            SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

             WHEN(DONT_CARE) DO;
              SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));
               WHEN(HIGH_IMPEDANCE,HIGH1_IMPEDANCE)
                OUT_VALUE(NEXT_ELMT,QSELECT) =
                                         OUT_VALUE(ELEMENT_NO,QSELECT);
               WHEN(CRITICAL_ONE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = HIGHO_IMPEDANCE;

               WHEN(LOGIC_ONE)
                OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;

               OTHERWISE RETURN(YES);

              END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;
```

```
         WHEN(LOGIC_ONE) DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
         END;

         OTHERWISE DO;
          OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
         END;

        END;
       END;

       OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WH,RETURN)');
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;
        TEST_FLAG(ELEMENT_NO) = TEST_FLAG(INLINE(ELEMENT_NO))
                   & (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRITO);
       END;

      END;
     END;

/*END
*/

/**TGWL**/
/* START        PROCESS LOW WIRED.
*/

     WHEN(LOW_WIRED) DO;
      SELECT(PREDIR(STACK_TOP));

       WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WL,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH CNTLINE  */

        NEXT_ELMT = CNTLINE(ELEMENT_NO);
        OTHER_ELMT = INLINE(ELEMENT_NO);

        IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
        THEN HIGH_VALUE = HIGHO_IMPEDANCE;
        ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
        THEN HIGH_VALUE = HIGH1_IMPEDANCE;

        SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));

         WHEN(CRITICAL_ZERO,CRITICAL_ONE) DO;
          PREDIR(STACK_TOP) = INDIR;
          SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

           WHEN(DONT_CARE) DO;
            IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                      OUT_VALUE(OTHER_ELMT,QSELECT)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                     OUT_VALUE(OTHER_ELMT,QSELECT));
           END;
          WHEN(LOGIC_ONE,CRITICAL_ONE)
           RETURN(YES);

          WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                                             HIGHO_IMPEDANCE) DO;
           IF OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE
           THEN RETURN(YES);
           ELSE
            CALL CONFLECT_WL_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
          END;

          WHEN(LOGIC_ZERO) DO;
           IF OUT_VALUE(OTHER_ELMT,QSELECT) = CRITICAL_ONE
           THEN RETURN(YES);
           ELSE CALL CONFLECT_WL_PROCESS(CRITICAL_ZERO);
          END;

          OTHERWISE RETURN(YES);

         END;
        END;

        WHEN(LOGIC_ZERO,LOGIC_ONE) DO;
         PREDIR(STACK_TOP) = INDIR;
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           IF OUT_VALUE(ELEMENT_NO,QSELECT) =
                                         OUT_VALUE(OTHER_ELMT,QSELECT)
           THEN DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
           END;
           ELSE DO;
            IF (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ONE
             &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE)
             | (OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
             &  OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ZERO)
            THEN DO;
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
             PREDIR(STACK_TOP) = CNTDIR;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
            END;
            ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,
                                  OUT_VALUE(OTHER_ELMT,QSELECT));
           END;
          END;

          WHEN(LOGIC_ONE,CRITICAL_ONE)
           RETURN(YES);

          WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),HIGH_IMPEDANCE,
                                             HIGHO_IMPEDANCE) DO;
           IF OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_ONE
           THEN RETURN(YES);
           ELSE
            CALL CONFLECT_WL_PROCESS(OUT_VALUE(OTHER_ELMT,QSELECT));
          END;

          WHEN(CRITICAL_ZERO) DO;
           IF OUT_VALUE(OTHER_ELMT,QSELECT) = LOGIC_ONE
           THEN RETURN(YES);
```

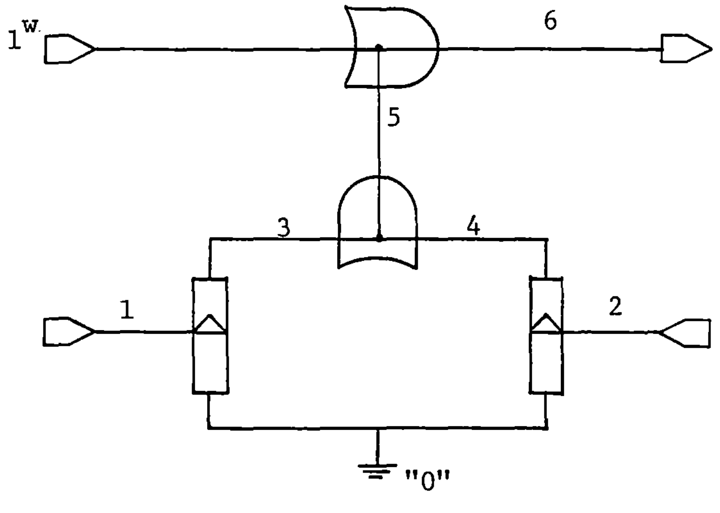
```
              ELSE CALL CONFLECT_WL_PROCESS(CRITICAL_ONE);
             END;

             OTHERWISE RETURN(YES);

            END;
           END;

           WHEN(DONT_CARE) DO;
            SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

             WHEN(DONT_CARE) DO;
              SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

               WHEN(HIGH_IMPEDANCE) DO;
                IF HIGH_VALUE = HIGH1_IMPEDANCE THEN DO;
                 NEXT_ELMT = OTHER_ELMT;
                 OUT_VALUE(ELEMENT_NO,QSELECT) = HIGH_VALUE;
                 PREDIR(STACK_TOP - 1) = AHEAD;
                END;
                ELSE PREDIR(STACK_TOP) = CNTDIR;
               END;

               WHEN(CRITICAL_ZERO) DO;
                IF (TEST_FLAG(NEXT_ELMT) & CRITO) = CRITO
                 & (TEST_FLAG(OTHER_ELMT) & CRITO) = NO_TEST
                THEN NEXT_ELMT = OTHER_ELMT;
                ELSE PREDIR(STACK_TOP) = CNTDIR;
               END;

               WHEN(CRITICAL_ONE,LOGIC_ONE,HIGH1_IMPEDANCE)
                NEXT_ELMT = OTHER_ELMT;

               OTHERWISE PREDIR(STACK_TOP) = CNTDIR;

              END;
              OUT_VALUE(NEXT_ELMT,QSELECT) =
                                         OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
             END;

             WHEN(CRITICAL_ZERO,HIGH_IMPEDANCE,HIGHO_IMPEDANCE) DO;
              PREDIR(STACK_TOP) = CNTDIR;
              IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= CRITICAL_ZERO
              THEN DO;
               OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO;
               PREDIR(STACK_TOP - 1) = AHEAD;
              END;
             END;

             WHEN(LOGIC_ZERO) DO;
              PREDIR(STACK_TOP) = CNTDIR;
              IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
              THEN DO;
               OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
              END;
              ELSE IF OUT_VALUE(ELEMENT_NO,QSELECT) ¬= LOGIC_ZERO
              THEN DO;
               OUT_VALUE(ELEMENT_NO,QSELECT) = LOGIC_ZERO;
               PREDIR(STACK_TOP - 1) = AHEAD;
              END;
             END;
```

```
              OTHERWISE RETURN(YES);

             END;
            END;

            OTHERWISE DO;
              /* WHEN INLINE IS ONE OF THE HIGH'S */
             PREDIR(STACK_TOP) = INDIR;
             SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

              WHEN(DONT_CARE) DO;
               SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

                WHEN(OUT_VALUE(OTHER_ELMT,QSELECT),CRITICAL_ZERO,
                                                   LOGIC_ZERO) DO;
                 OUT_VALUE(NEXT_ELMT,QSELECT) =
                                            OUT_VALUE(ELEMENT_NO,QSELECT);
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                END;

                WHEN(HIGH_IMPEDANCE) DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH1_IMPEDANCE
                 THEN CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH1_IMPEDANCE);
                 ELSE DO;
                  OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
                 END;
                END;

                OTHERWISE DO;
                 IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH1_IMPEDANCE
                 THEN CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH1_IMPEDANCE);
                 ELSE CALL CONFLECT_GO(HIGH_IMPEDANCE,HIGH_IMPEDANCE);
                END;

               END;
              END;

              WHEN(CRITICAL_ZERO)
               CALL CONFLECT_WL_PROCESS(CRITICAL_ZERO);

              WHEN(LOGIC_ZERO) DO;
               IF OUT_VALUE(ELEMENT_NO,QSELECT) = CRITICAL_ZERO
               THEN DO;
                OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
               END;
               ELSE CALL CONFLECT_WL_PROCESS(LOGIC_ZERO);
              END;

              WHEN(HIGH_IMPEDANCE) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH1_IMPEDANCE
               THEN CALL CONFLECT_WL_PROCESS(HIGH1_IMPEDANCE);
               ELSE CALL CONFLECT_WL_PROCESS(HIGH_IMPEDANCE);
              END;

              WHEN(HIGHO_IMPEDANCE) DO;
               IF OUT_VALUE(OTHER_ELMT,QSELECT) = HIGH_VALUE
               THEN CALL CONFLECT_WL_PROCESS(HIGH_VALUE);
               ELSE CALL CONFLECT_WL_PROCESS(HIGH_IMPEDANCE);
              END;
            OTHERWISE RETURN(YES);

           END;
          END;

         END;
        END;

        WHEN(CNTDIR) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WL,CNTDIR)');
          /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */
         PREDIR(STACK_TOP) = INDIR;
         NEXT_ELMT = INLINE(ELEMENT_NO);
         OTHER_ELMT = CNTLINE(ELEMENT_NO);
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           SELECT(OUT_VALUE(OTHER_ELMT,QSELECT));
            WHEN(HIGH_IMPEDANCE,HIGHO_IMPEDANCE)
             OUT_VALUE(NEXT_ELMT,QSELECT) =
                                     OUT_VALUE(ELEMENT_NO,QSELECT);
            WHEN(CRITICAL_ZERO)
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH1_IMPEDANCE;

            WHEN(LOGIC_ZERO)
             OUT_VALUE(NEXT_ELMT,QSELECT) = HIGH_IMPEDANCE;

            OTHERWISE RETURN(YES);

           END;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
          END;

          WHEN(LOGIC_ONE) DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ONE;
%INCLUDE TGCNE;
          END;

          OTHERWISE DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = CRITICAL_ZERO;
%INCLUDE TGCNE;
          END;

         END;
        END;

        OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WL,RETURN)');
          /* PROCESS RETURN FROM THIS ELEMENT */
         STACK_TOP = STACK_TOP - 1;
         TEST_FLAG(ELEMENT_NO) = TEST_FLAG(INLINE(ELEMENT_NO))
                    & (TEST_FLAG(CNTLINE(ELEMENT_NO)) | CRIT1);
        END;

       END;
      END;

/*END
*/

/**TGWP**/
/* START        PROCESS PULL UP/DOWN  WIRED.
```

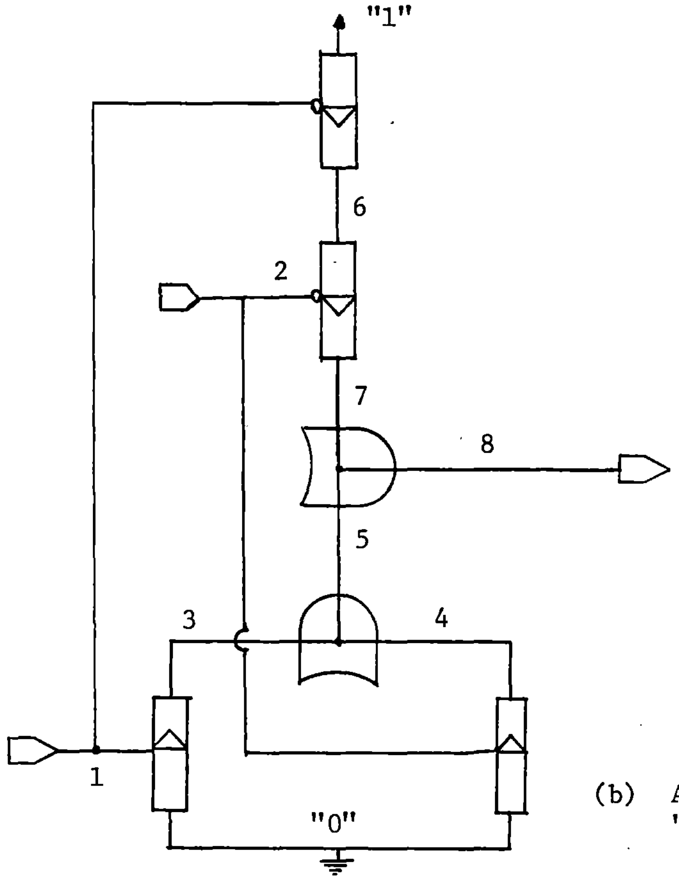
```
*/

      WHEN(PULL_UD_WIRED) DO;
       NEXT_ELMT = INLINE(ELEMENT_NO);
       OTHER_ELMT = CNTLINE(ELEMENT_NO);
       SELECT(PREDIR(STACK_TOP));

        WHEN(AHEAD) DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WP,AHEAD)');
         /* PROCESS CALLING NEXT ELEMENT THROUGH INLINE  */
         PREDIR(STACK_TOP) = INDIR;
         SELECT(OUT_VALUE(ELEMENT_NO,QSELECT));

          WHEN(CRITICAL_ZERO) DO;
           IF OTHER_ELMT = WEAK_ONE
           THEN LOGIC_VALUE = CRITICAL_ZERO;
           ELSE LOGIC_VALUE = HIGH1_IMPEDANCE;
          END;

          WHEN(CRITICAL_ONE) DO;
           IF OTHER_ELMT = WEAK_ONE
           THEN LOGIC_VALUE = HIGHO_IMPEDANCE;
           ELSE LOGIC_VALUE = CRITICAL_ONE;
          END;

          WHEN(LOGIC_ZERO) DO;
           IF OTHER_ELMT = WEAK_ONE
           THEN LOGIC_VALUE = LOGIC_ZERO;
           ELSE LOGIC_VALUE = HIGH_IMPEDANCE;
          END;

          WHEN(LOGIC_ONE) DO;
           IF OTHER_ELMT = WEAK_ONE
           THEN LOGIC_VALUE = HIGH_IMPEDANCE;
           ELSE LOGIC_VALUE = LOGIC_ONE;
          END;

          OTHERWISE RETURN(YES);

         END;
         SELECT(OUT_VALUE(NEXT_ELMT,QSELECT));

          WHEN(DONT_CARE) DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_VALUE;
%INCLUDE TGCNE;
   QUEUE_TOP = QUEUE_TOP + 1; QUEUE(QUEUE_TOP,QSELECT) = NEXT_ELMT;
          END;

          WHEN(LOGIC_VALUE) DO;
           STACK_TOP = STACK_TOP - 1;
           TEST_FLAG(ELEMENT_NO) =
            TEST_FLAG(ELEMENT_NO) | TEST_FLAG(NEXT_ELMT);
          END;

          WHEN(LOGIC_ONE) DO;
           IF LOGIC_VALUE = CRITICAL_ONE THEN DO;
            OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_VALUE;
%INCLUDE TGCNE;
           END;
           ELSE CALL CONFLECT_WP_RETURN(LOGIC_ONE);
          END;

          WHEN(LOGIC_ZERO) DO;
           IF LOGIC_VALUE = CRITICAL_ZERO THEN DO;
           OUT_VALUE(NEXT_ELMT,QSELECT) = LOGIC_VALUE;
%INCLUDE TGCNE;
          END;
          ELSE CALL CONFLECT_WP_RETURN(LOGIC_ZERO);
         END;

         WHEN(HIGHO_IMPEDANCE,CRITICAL_ONE)
          CALL CONFLECT_WP_RETURN(CRITICAL_ONE);

         WHEN(HIGH1_IMPEDANCE,CRITICAL_ZERO)
          CALL CONFLECT_WP_RETURN(CRITICAL_ZERO);

         OTHERWISE DO;
          IF OTHER_ELMT = WEAK_ONE
          THEN CALL CONFLECT_WP_RETURN(LOGIC_ONE);
          ELSE CALL CONFLECT_WP_RETURN(LOGIC_ZERO);
         END;

        END;
       END;

       OTHERWISE DO;
IF TRACE_FLAG THEN CALL TRACE('TGJOPATH (WP,RETURN)');
         /* PROCESS RETURN FROM THIS ELEMENT */
        STACK_TOP = STACK_TOP - 1;
        TEST_FLAG(ELEMENT_NO) =
         TEST_FLAG(ELEMENT_NO) | TEST_FLAG(NEXT_ELMT);
       END;

      END;
     END;

/*END
*/
```

APPENDIX B

EXAMPLES



(a) A 2-Input NMOS "NOR" Gate



(b) A 2-Input CMOS "NOR" Gate

Figure 36. The wGS Representation of 2-Input "NOR" Gates

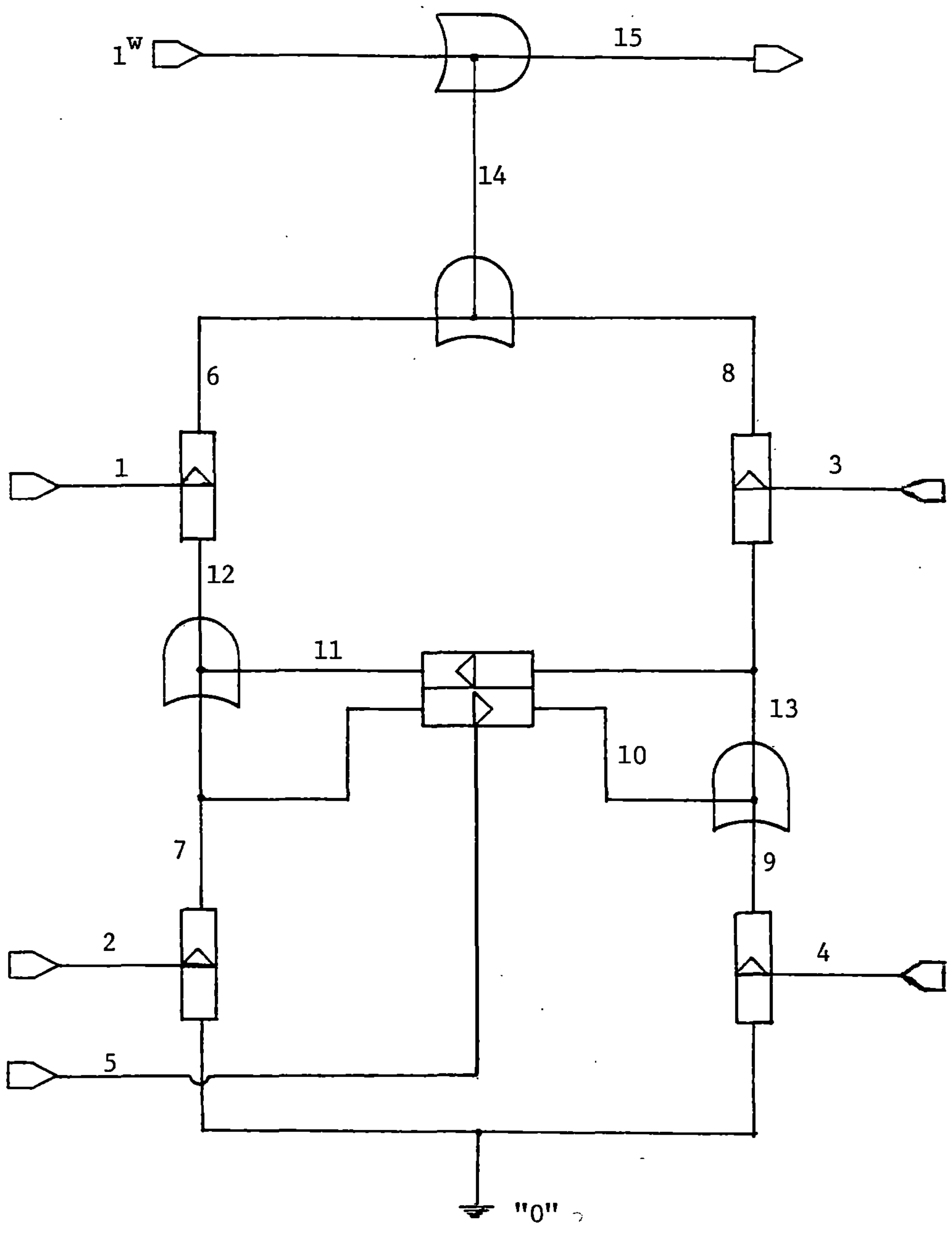


Figure 37. The wGS Representation of a Bridge Circuit

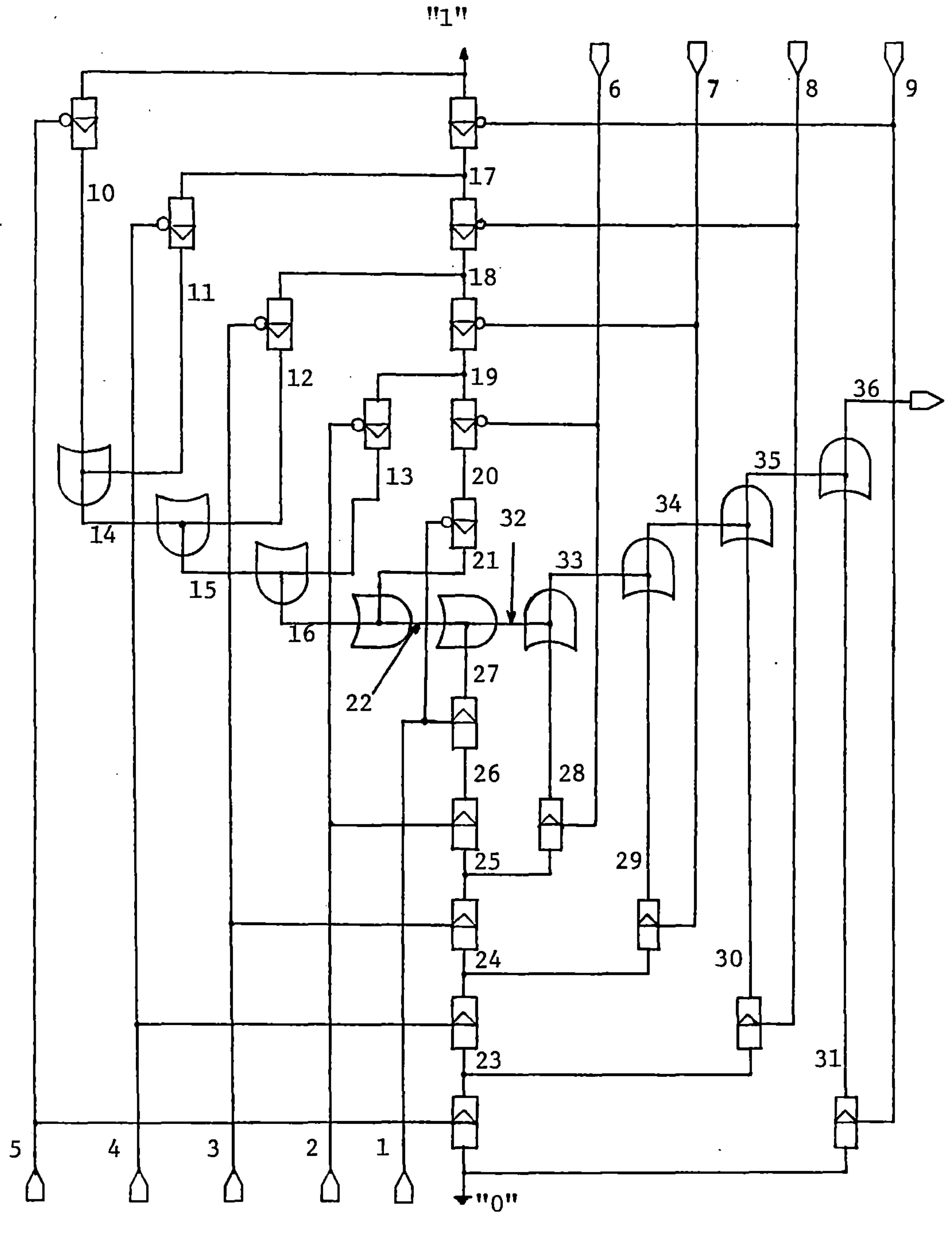


Figure 38. The wGS Representation of a SCLG Circuit

The wGS input network representation for Figure 33

```
MAX 21
PI 5 5
PI 1 1
PI 2 2
PI 4 4
PI 3 3
PI 6 6
IP 7 0 5
WU 8 7
IP 9 0 6
WU 10 9
PS 11 1 8
PS 12 2 8
PS 13 3 5
PS 14 4 5
PS 15 12 6
PS 16 14 6
PS 17 11 10
PS 18 13 10
WG 19 15 17
WG 20 16 18
WG 21 20 19
PO 1 21
```

Generated tests by computer program, listed in Appendix A, for Figure 33

```
       1         0        21 0 X11011
       2         1        21 1 X00111
       3         0        21 0 1X0110
       4         3        21 1 0X1010
       5         0        21 0 10X101
       6         5        21 1 01X001
       7         0        21 0 011X00
       8         7        21 1 100X00
```

The wGS input network representation for Figure 36(a)

```
MAX 6
PI   1   1
PI   2   2
IP   3   0   1
IP   4   0   2
WG   5   4   3
WU   6   5
PO   1   6
```

Generated tests by computer program, listed in Appendix A, for Figure 36(a)

```
       1         0         6 0 01
       2         0         6 1 00
       3         0         6 0 10
```

The wGS input network representation for Figure 36(b)

```
MAX   8
PI    1   1
PI   2  2
IP   3  0  1
IP   4  0  2
WG   5  4  3
IN   6  1  1
NS   7  6  2
WC   8  5  7
PO   1  8
```

Generated tests by computer program, listed in Appendix A, for Figure 36(b)

```
        0        1        8 X 00
        1        0        8 0 01
        2        0        8 1 00
        0        3        8 X 00
        3        0        8 0 10
```

The wGS input network representation for Figure 37

```
MAX 15
PI   1  1
PI   2  2
PI   3  3
PI   4  4
PI   5  5
PS   6 12  1
IP   7  0  2
PS   8 13  3
IP   9  0  4
BP  10  7  5
BP  11 13  0
WG  12 11  7
WG  13 10  9
WG  14  8  6
WU  15 14
PO   1 15
```

Generated tests by computer program, listed in Appendix A, for Figure 37

```
        1        0        15 0 01101
        2        0        15 1 01010
        3        0        15 0 0X110
        4        0        15 1 10010
        5        0        15 0 10011
        6        0        15 1 10101
```

The wGS input network representation for Figure 38

```
MAX 36
PI   1  1
PI   2  2
PI   3  3
PI   4  4
PI   5  5
PI   6  6
PI   7  7
PI   8  8
PI   9  9
```

```
IN 10  1  5
NS 11 17  4
NS 12 18  3
NS 13 19  2
WG 14 10 11
WG 15 14 12
WG 16 15 13
IN 17  1  9
NS 18 17  8
NS 19 18  7
NS 20 19  6
NS 21 20  1
WG 22 16 21
IP 23  0  5
PS 24 23  4
PS 25 24  3
PS 26 25  2
PS 27 26  1
PS 28 25  6
PS 29 24  7
PS 30 23  8
IP 31  0  9
WC 32 27  22
WL 33 32 28
WL 34 33 29
WL 35 34 30
WL 36 35 31
PO  1 36
```

Generated tests by computer program, listed in Appendix A, for Figure 38

```
        1         0        36 0 111110000
        0         2        36 X 11111XXXX
        2         0        36 1 111100000
        3         0        36 0 011111000
        0         4        36 X 11111XXXX
        4         0        36 1 111010000
        5         0        36 0 001110100
        0         6        36 X 11111XXXX
        6         0        36 1 110110000
        7         0        36 0 000110010
        0         8        36 X 11111XXXX
        8         0        36 1 101110000
        9         0        36 0 000010001
        0        10        36 X 11111XXXX
       10         0        36 1 011110000
```

VITA

Mohammad Taghi Mostafavi

Candidate for the Degree of

Doctor of Philosophy

Thesis: TEST GENERATION AT THE TRANSISTOR LEVEL FOR MOS COMBINATIONAL LOGIC CIRCUITS

Major Field: Electrical Engineering

Minor Field: Computer and Information Sciences

Biographical:

Personal Data: Born in Busheir, Iran, June 18, 1951, the son of Asghar and Bozorg. Married to Zahra Bahrani on March 18, 1977. Has a four-year-old son named Behrooz.

Education: Graduated from Seadat High School, Busheir, Iran, in June, 1969; received Bachelor of Science Degree in Computer and Information Science from Oklahoma State University in December 1980; received Master of Science in Electrical Engineering from Oklahoma State University in May, 1982; completed requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering with a minor in Computer and Information Sciences at Oklahoma State University in May, 1986.

Professional Experience: Computer Programmer, Department of Agricultural Economics, Oklahoma State University, September, 1980 to January, 1981; Research Assistant, Department of Agricultural Economics, Oklahoma State University, January, 1981 to September 1981; Teaching Assistant, Department of Electrical and Computer Engineering, Oklahoma State University, August, 1984 to July, 1985; System Analyst Programmer, Department of Agricultural Economics, Oklahoma State University, September, 1981, to present.