FAULT DETECTION AND LOCATION IN MODULAR TREES

by

Lionel C. C. Shih

Thesis submitted to the Graduate Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

F. G. Gray, Chairman

R. A. Thompson

H. F. VanLandingham

May, 1976

Blacksburg, Virginia

ACKNOWLEDGMENTS

The author wishes to thank Dr. F. Gail Gray, Richard A. Thompson and Hugh F. VanLandingham for serving on his advisory committee. He is greatly indebted to his committee chairman, Dr. F. Gail Gray, for the latter's generosity in giving of his time, guidance, and encouragement during the past nine months. He appreciates the inspiration Dr. Richard A. Thompson gave him at the beginning to undertake a thesis project.

And, to the National Science Foundation, he is grateful for funding this research under grant DCR75-06543.

The author owes a great debt of appreciation to his wife, , for the long hours she spent typing the final manuscript in time to meet the required deadline. And he especially wants to express his thanks to his parents for their continued faith in him and for their encouragement.

TABLE OF CONTENTS

1. INTRODUCTION

In order to construct a large multi-functional digital system using iterative arrays, we would like to have in the array cells which are functionally complete and which can be dynamically reconfigured to perform alternate functions. Since the modular tree is functionally complete and its behavior is totally determined by programming its accessible control leads [6], it is a promising candidate for this application. However, another important parameter, reliability, in the design of a digital system, should never be ignored, especially in a large system. Introducing redundancy in the system is one way to improve reliability, but system reconfigurability may be more powerful and desirable particularly where manual service is not available. Error detection in a system with redundant components can be done by voting. However, in a reconfigurable system, the system itself must be able to detect errors if they exist, locate them, and make a decision to reconfigure itself properly. In such a system, fault detection and location of each fault is obviously desirable, but, the ability to reconfigure the faulty component for masking faults is also important. Again, the modular tree structure proves itself superior in this respect for the design of systems of array structures. Fault masking by reconfiguration can be easily implemented. [2,9]

Fig. 1a shows a modular tree structure for realization of any combinational function of n primary input variables $(x_1,.....,x_n)$.

By applying suitable patterns of ZEROS and ONES on the control leads ($C_0, \ldots\ldots C_{2^n-1}$) of the tree, any combinational function of n primary input variables can be realized. (An ordered set of values for C_0, $C_1, \ldots\ldots\ldots C_{2^n-1}$ will be called a control pattern). In fact there are 2^n control leads in the tree and there exists a one-to one correspondence between the 2^{2^n} control patterns that can be applied to the control leads and the 2^{2^n} combinational functions of n primary input variables. And that is why the tree structure is claimed functionally complete. If all those control leads are accessible, then, obviously, the change of function is done by just reassigning (reprogramming) the ONES and ZEROS on the control leads. The basic cell structure with m inputs, shown in Fig. 1b, is a two level network with 2^m AND gates on the first level and a single OR gate on the second level. One of the m+1 inputs of each AND gate is the control lead; all the other inputs are from the m primary inputs (possibly inverted). Primary inputs are common to each AND gate in a cell, but each AND gate has a separate control lead. For instance, in Fig. 4, x_1, x_2 are primary inputs, C_0 to gate 0, C_1 to gate 1, etc. are control leads.

By simple modification of both the cell and the modular tree, as shown in Fig. 2a and 2b, the modified tree structure is able to realize any binary definite machine with order of definiteness d [7]. Freidman [8] has shown that any sequential machine can be realized by a sequential network with a single feedback wire. Using this result, Arnold [7] gave a universal structure as shown in Fig. 3

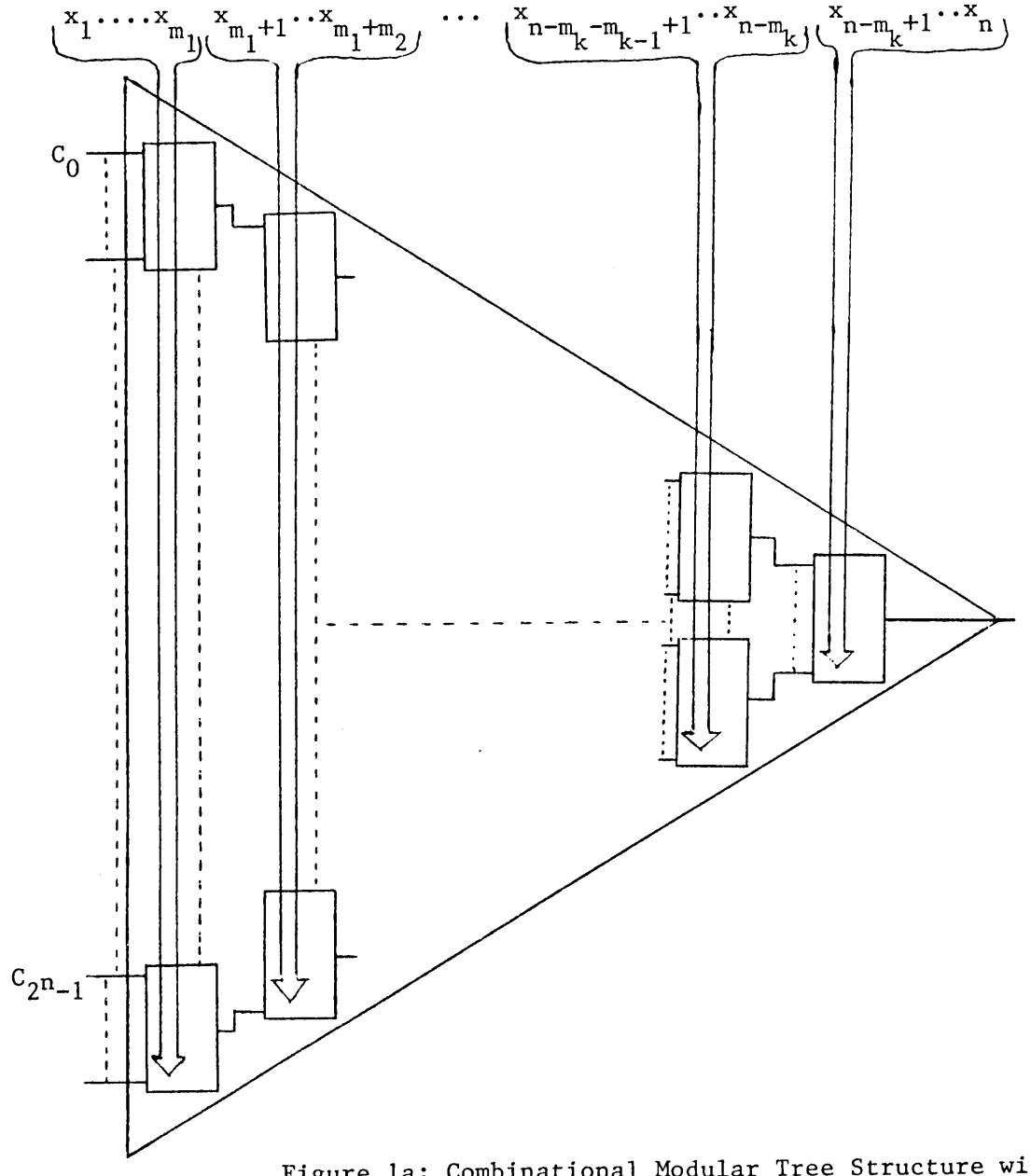

Figure 1a: Combinational Modular Tree Structure with $n = \sum_{h=1}^{k} m_h$ Input Variables

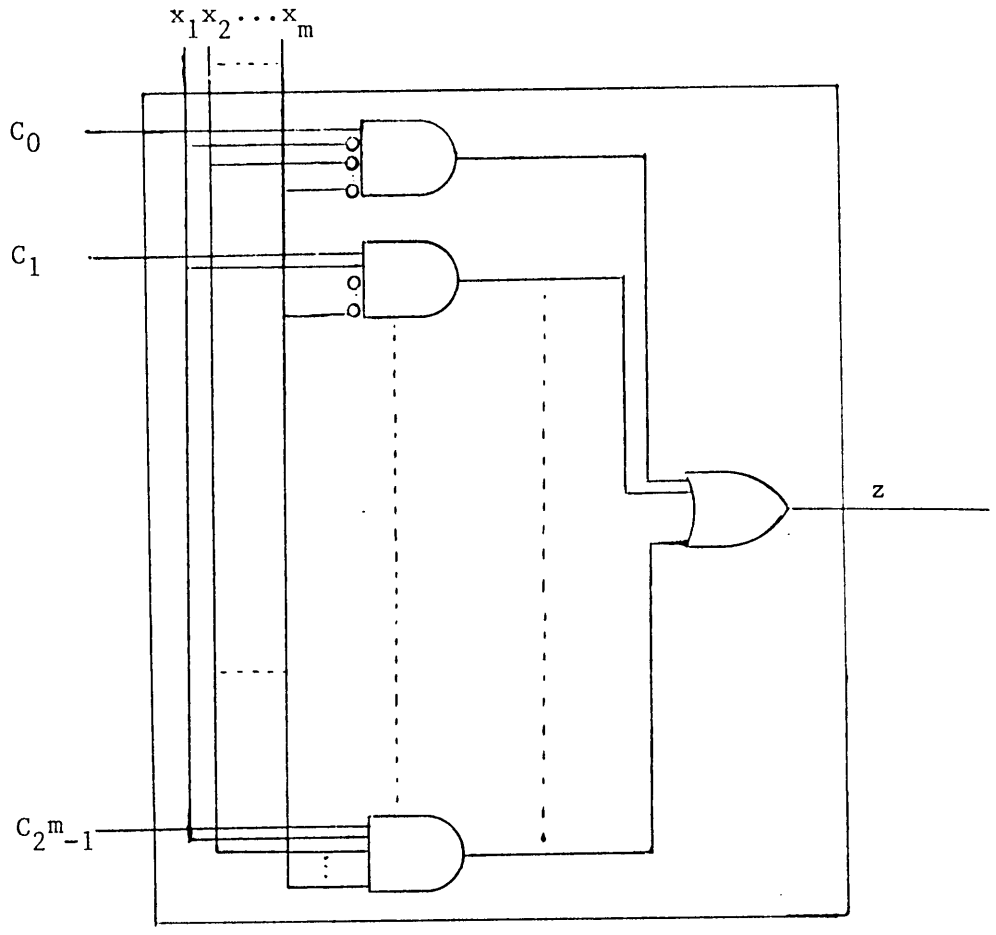

Figure 1b: The Structure of a Cell used in Figure 1a

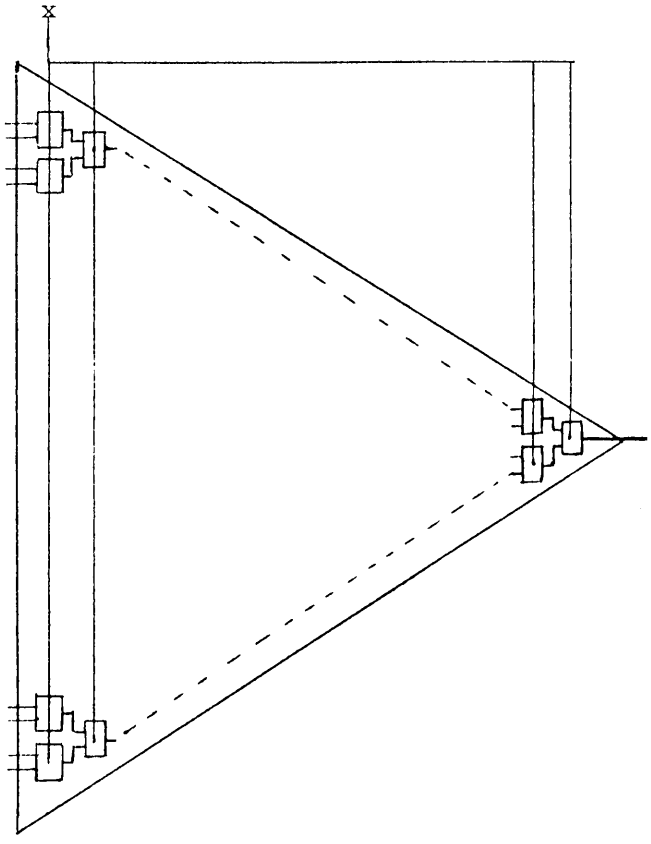

Figure 2a: Modular Tree Structure for Realizing Any Binary Definite Machine

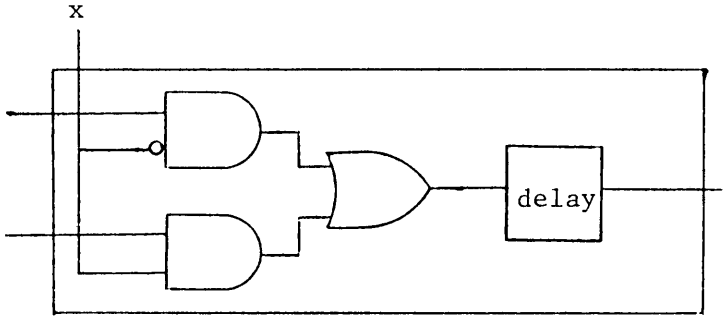

Figure 2b: Details of Cell Structure used in Figure 2a

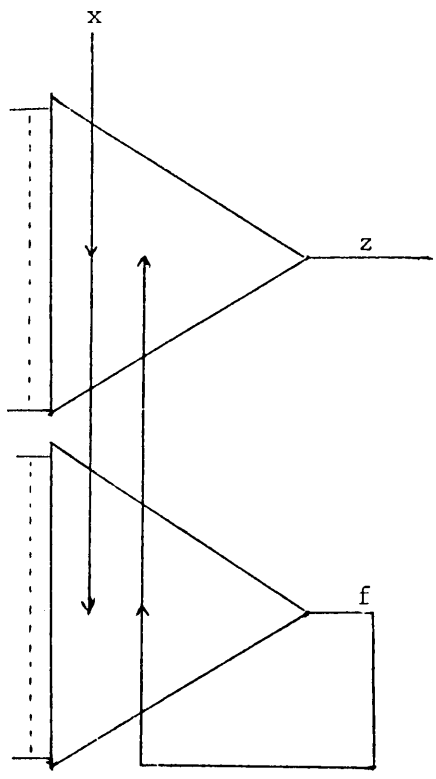

Figure 3a: Modular Tree Structure for Realizing Arbitrary Sequential Machines

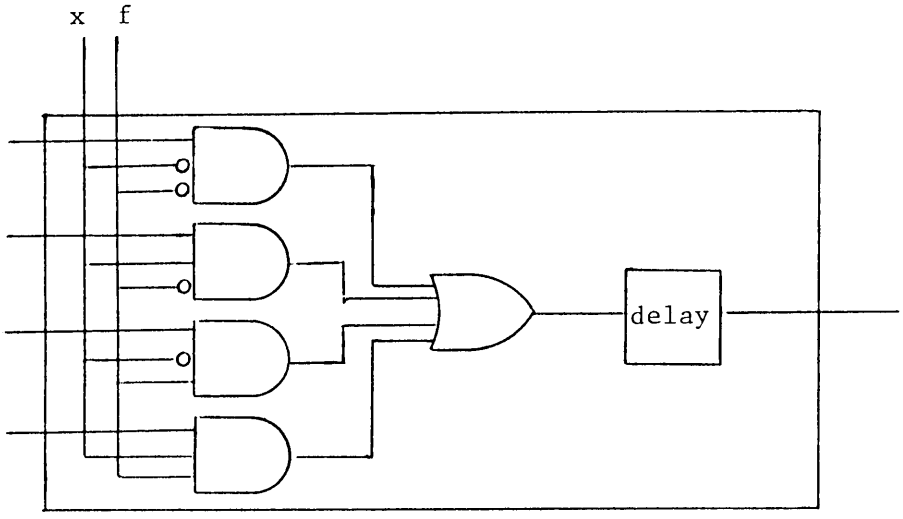

Figure 3b: Details of Cell Structure used in Figure 3a

for realization of any binary sequential machine.

Because of so many interesting features possessed by the modular tree structure, extensive work on the fault detection problem in this structure has been done. However, before reviewing the previous work done on fault detection in a tree and introducing the work done in this thesis, those faults which are going to be discussed in the following chapters either for detection or location are defined to avoid confusion. Positive logic is going to be used consistently throughout this thesis.

A stuck-at-fault (s-a-f) on a wire will cause the wire to be either logic 0 (s-a-0) or logic 1 (s-a-1), for all time.

A short circuit (SC) fault will cause all those wires involved to stay at logic 0, unless all of them are pulled high (logic 1) simultaneously.

A broken and short circuit (BASC) fault will cause the broken wire to follow the wire to which it is shorted.

The faults are illustrated in Fig. 5.

Cioffi & Fiorillo [11] have considered fault detection under unrestricted fault assumptions in combinational trees of the type shown in Fig. 1a. The test requires all the control leads of the tree to be accessible. If the tree in question is required to realize only a particular function, then the value assigned to each control lead should be constant logical ZERO or ONE which can be done during fabrication. Under this condition, it is necessary only to

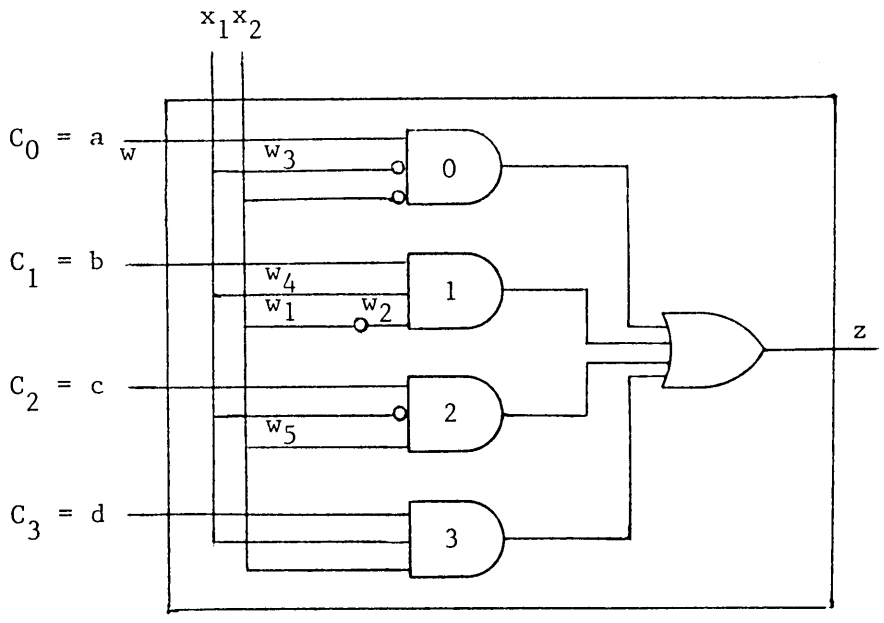

Figure 4: Details of a Cell with Two Input Variables

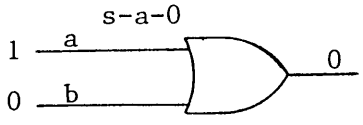

Figure 5a: Example of s-a-f (wire 'a' s-a-0)

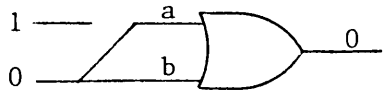

Figure 5b: Example of BASC fault (wire 'a' BASC wire'b')

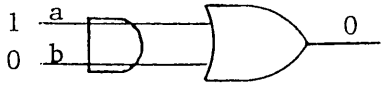

Figure 5c: Example of SC fault (wire 'a' SC wire 'b')

test whether the tree realizes the particular function or not. Ramos and Smith [12,13] consider such a fault detection problem in trees shown in Fig. 2a and 3 with s-a-f's only on the control path (a control path is a path from a control lead to the output of the tree).

An algorithm for detecting multiple s-a-f's in a modular tree is given by Prasad and Gray [1,2]. In fact, the whole procedure only tests whether the tree can realize two special functions. In other words, if assuming only s-a-f's may happen and the tree with n input variables passes the test, the tree can realize any one of the 2^{2^n} combinational functions of n binary variables. Fault detection of another fault model, relaxed fault model in a tree, is also considered in [1,2] and a testing algorithm is given there. Both of the two algorithms have been modified for definite trees.

But BASC fault and SC fault are more difficult to analyze than s-a-f's. Even though a bridging fault model (equivalent to SC fault model) has been considered by Mei [14], the discussion is not in connection with modular trees. At this time, detection of BASC and SC faults is discussed under certain restrictions in Chapter 4.

In fact, the main theme of this thesis is on the s-a-f location problem in a modular tree which is an untouched topic also. The location problem of s-a-f's is studied in Chapter 2 and 3 of this thesis.

2. FAULT LOCATION IN CELLS

In this chapter, distinguishability among s-s-f's in a modular tree is briefly studied from the viewpoint of both physical constraints and utilization of the faulty tree[1,9]. Then, important tools (control patterns) are introduced and their particular properties useful for fault location are discussed. Even though, a single cell is analyzed in this chapter, the special structure of the tree enables the results to be applied to the whole modular tree with slight modification.

2.1 Distinguishability of s-a-f's in a Cell

Definitions:

f_0 fault in a gate: any s-a-f which is indistinguishable from a s-a-0 on some gate input.

f_1 fault in a gate: any s-a-f which is indistinguishable from a s-a-1 on some gate input.

For example, if there is an inverter between a primary input and an AND gate input, an f_0 (f_1) may be caused by a s-a-1 (or s-a-0) on the primary input or a s-a-0 (or s-a-1) on the output of the inverter. Referring to Fig. 4, w_1 s-a-0 or w_2 s-a-1 are both f_1 faults on gate 1 and w_1 s-a-1 or w_2 s-a-0 are both f_0 faults on gate 1. Since w_1 s-a-1(0) and w_2 s-a-0(1) are indistinguishable, we do not need to consider them separately. Similary, all f_0 faults in an AND gate are indistinguishable from each other, and all f_1 faults

in an OR gate are indistinguishable. The former is equivalent to s-a-0 on the output of the AND gate, the latter to s-a-1 on the output of the OR gate. Therefore, if an AND gate (OR gate) has any f_0 (f_1) fault, it will be able to perform only the constant function 0(1). So from the viewpoint of faulty tree utilization, indistinguishable faults present no problems. Once the faulty gate is located, possible reconfigurations of the faulty tree are uniquely defined no matter which wire in that gate causes the error.

Nevertheless, there are many faults which we do want to locate to the wire level because they lead to different tree utilizations.

To utilize a faulty tree with a s-a-0 (s-a-1) fault on some control lead, a control pattern must be applied with 0(1) on the faulty lead. But, f_1 faults on primary inputs to an AND gate lead to different tree utilization[1,2]. The reason for this is illustrated briefly in the following example.

Example 2-1: Refer to Fig. 4

$z = a\bar{x}_2\bar{x}_1 + b\bar{x}_2x_1 + cx_2\bar{x}_1 + dx_2x_1$ is the correct output function.

If an f_1 fault occurs on the x_2 input of gate 1,

then $z' = a\bar{x}_2\bar{x}_1 + bx_1 + cx_2\bar{x}_1 + dx_2x_1$

$= a\bar{x}_2\bar{x}_1 + b(x_2+\bar{x}_2)x_1 + cx_2\bar{x}_1 + dx_2x_1$

$= a\bar{x}_2\bar{x}_1 + b\bar{x}_2x_1 + cx_2\bar{x}_1 + (b+d)x_2x_1$

For this example, if b=0 or b=d=1 then z=z'. Therefore the tree can be utilized to perform any function for which one of these two conditions is satisfied. Specific conditions on control patterns

that satisfy these conditions can be found in [2,9].

Due to different tree utilizations, f_1 faults on an AND gate must be located to the precise gate input. However, in Fig. 4, a s-a-1 fault on w_2 is still equivalent to a s-a-0 fault on w_1.

Some terminology useful for later discussion is stated below.

2.2 Properties of (i,k)- and (i,k)'-patterns

Definition:

A binary string of length 2^k ($k \varepsilon N$, N is the set of positive integers) consisting of 2^{k-1} 0's (1's) followed by 2^{k-1} 1's (0's) is called a k-pattern.

Example 2-2: 1-patterns: 01, 10 2-patterns: 0011, 1100

Definition:

Representing the elements of the k-pattern with binary codes from 0 to 2^{k-1}, the label of each such binary code is 0 if the value of the corresponding pattern element is 0, and is 1 if the value of the corresponding pattern element is 1.

Example 2-3:

Pattern	Binary Code	
1	00	
1	01	The binary codes 00 and 01
0	10	have 1-labels, 10 and 11
0	11	have 0-labels.

Definition:

An (i,k)-pattern of length 2^{i+k} is constructed by concatenating a k-pattern recursively with itself until the string has the length 2^{i+k}. (In fact, each k-pattern is a (0,k)-pattern.)

Example 2-4:

(0,1)-patterns: 01, 10 (1,1)-patterns: 0101, 1010

(2,1)-patterns: 01010101, 10101010

Definition:

An (i,k)'-pattern of length 2^{i+k} is constructed from an (i-1,k)'-pattern by complementing it and concatenating with the original pattern. (A (0,k)'-pattern is a k-pattern.)

Example 2-5:

(0,1)'-patterns: 01, 10 (1,1)'-patterns: 0110, 1001

(2,1)'-patterns: 01101001, 10010110

Definition:

For an (i,k)'-pattern, i+1 is the index number of that pattern.

The following two lemmas illustrate some interesting properties possessed by (i,k) and (i,k)'-patterns respectively.

Lemma 1: In an (i,k)-pattern ($k \in N$), each 0(1)-label code c has exactly one other 1(0)-label code which is Hamming distance ONE from c and the difference occurs at the kth bit.

Proof: For any (0,k)-pattern ($k \in N$), the binary number used to represent the elements of the pattern is k-bits wide. Obviously, the kth bit (most significient) will be 0 in the first half of the

representation, and 1 in the second half. Two codes from different halves are HD ONE from each other if they are different only in the kth bit. Obviously, each code in a k-pattern always has exactly one other code which is HD ONE from it at the kth bit. Also, the two halves of the representation are differently labeled. The Lemma is proved for (0,k)-patterns.

Induction basis: the argument above proves the Lemma for (0,k)-patterns.

Induction hypothesis: assumed true for (i-1,k)-patterns

Now to show it is true for (i,k)-patterns:

Any (i,k)-pattern is constructed by cancatenating two identical (i-1,k)patterns together. So, the binary representation of the (i,k)-pattern will consist of (i+k)-bit codes instead of (i+k-1)-bit codes. Those codes representing the first (i-1,k)-pattern always has 0 on its most significant bit but those codes representing the second (i-1,k)-pattern will have 1 on that bit. There obviously can be only one code in the first (i-1,k)-pattern HD ONE from each code in the second pattern, and these codes will be identical except for the (i+k)-bit. This means they represent the same bit in the two (i-1,k)-patterns and hence have the same label; any other codes HD ONE apart must both represent bits in the same (i-1,k)-pattern, and hence are covered by the induction hypothesis. Therefore, for each code c in (i,k)-pattern, there is exactly one code which is labeled differently and HD1 from code c.

Lemma 2: In an (i,k)'-pattern, for each 0(1)-label code c, the index number, (i+1), gives the exact number of 1(0)-label codes which are Hamming Distance ONE from c.

Proof: Lemma will be proved by induction on i.

Basis: it is true when i is equal to 0 by the proof of Lemma 1.

Induction hypothesis: assumed true for i-1

Now to show it is true for i.

Same argument of the proof of lemma 1 can be followed except that those two codes occupying corresponding positions in two patterns are differently labeled. Therfore, each code in an (i,k)-pattern will have one more code with different label and HD ONE from it than those in ((i-1),k)-pattern have. By induction hypothesis, the index number of ((i-1),k)'-pattern, that is i, gives the number of such codes. Therefore, for (i,k)'-pattern, i+1 is the number of different-labeled HD1 codes of code c.

Example 2-6:

(1,1)-pattern	(1,1)'-pattern	code
0	0	00
1	1	01
0	1	10
1	0	11

In this example, 00 is a 0-label code in both patterns. Codes 01 and 10 are the only codes Hamming distance ONE from 00. As predicted by Lemma 1, there is exactly one code 01 with a 1-label that is HD ONE from 00 in the (1,1)-pattern. Lemma 2 says there should be two 1-labeled codes HD ONE from 00 in the (1,1)'-pattern.

2.3 Unique Detection Condition of s-a-f's in a Cell

In addition to the terminology already introduced, the notion of unique detection condition and some useful notation must be presented before handling the problem of fault location.

Lemma 3: If $C = (C_0, C_1, \ldots\ldots, C_{2^n-1})$ is the control pattern needed to realize the function $f(x_1, x_2, \ldots\ldots, x_n)$ using the module shown in Fig. 1b, then $f(\alpha_1, \alpha_2, \ldots\ldots, \alpha_n) = C_j$, for all j where $[j]_{10} = [\alpha_n \alpha_{n-1} \ldots\ldots \alpha_2 \alpha_1]_2$ (read j base 10 equals $\alpha_n \alpha_{n-1} \ldots \alpha_2 \alpha_1$ base 2). The proof of this lemma appears in [2].

Example 2-7: Refer to Fig. 4.

If $C = (C_0, C_1, C_2, C_3) = (a, b, c, d)$ where $C_0=C_1=C_3=0$ and $C_2=1$, then input 00, 01 or 11 will produce output 0 from the module since the value of the corresponding control leads C_0, C_1, and C_3 are 0. Similarly, if we input 10 to the module, the output will be 1 since C_2 equals 1.

Definition:

For each j where $[j]_{10} = [\alpha_n \cdots \alpha_k \cdots \alpha_1]_2$, the number j_k^* is defined as $[j_k^*]_{10} = [\alpha_n \cdots \overline{\alpha}_k \cdots \alpha_1]_2$ for each k in the range $1 \leq k \leq n$.

Example 2-8: Refer to Fig. 4.

Let j = 2, then $[j]_{10} = [2]_{10} = [10]_2$

and $[j_1^*]_{10} = [2_1^*]_{10} = [11]_2 = [3]_{10}$

and $[j_2^*]_{10} = [2_2^*]_{10} = [00]_2 = [0]_{10}$

Let j = 3, then $[j]_{10} = [3]_{10} = [11]_2$

and $[j_1^*]_{10} = [3_1^*]_{10} = [10]_2 = [2]_{10}$

and $[j_2^*]_{10} = [3_2^*]_{10} = [01]_2 = [1]_{10}$

Summarizing, $2_1^* = 3$, $2_2^* = 0$, $3_1^* = 2$, and $3_2^* = 1$.

The notion of sensitizing a s-a-f is introduced briefly below. If w_i is stuck at 0, the fault will be sensitized if and only if a test assigns value 1 on w_i and a path is provided to propagate the error to the output. Because of Lemma 3, a sensitized path always exists for each s-a-f, so the basic problem is assigning the faulty wire a value opposite to the s-a-f.

Unique detection condition is the next topic to be discussed and gives the theoretical basis for choosing appropriate control patterns to detect and locate s-a-f's in a cell in a tree structure.

Lemma 4: In a single cell, as in Fig. 1b, an f_1 fault on the x_k input wire of the jth AND gate where $1 \leq k$, and $0 \leq j \leq 2^n-1$, can be detected if and only if $C_j = 1$, $C_{j_k^*} = 0$ and $(\alpha_1, \alpha_2, \ldots\ldots, \alpha_n)$ is input where $[\alpha_n \alpha_{n-1} \ldots\ldots \alpha_1]_2 = [j_k^*]_{10}$.

Proof: Let $[j]_{10} = [\alpha_n \ldots \alpha_1]_2$ and $[j_k^*]_{10} = [\alpha_n \ldots \bar{\alpha}_k \ldots\ldots \alpha_1]_2$.

Sufficiency:

Since $C_{j_k^*} = 0$ and input is $[j_k^*]_{10}$ the output of the cell should be 0 by Lemma 3. However, the f_1 fault on the input wire of kth input variable to the jth gate together with $C_j = 1$ will cause the cell output 1 when input is $[j_k^*]_{10}$. Error occurs.

Necessity:

Suppose there is another test that will work and let it be:

$C_j = x$, $C_{j_k^*} = y$ and $[\tau_n \ldots\ldots \tau_1]_2 = [\ell]_{10}$ where $[\ell]_{10} \neq [j_k^*]_{10}$. If $[\ell]_{10} = [j]_{10}$, then output will follow the value of C_j and no

error will occur. This is contradiction.

If $[\ell]_{10} \neq [j]_{10}$ and since $[\ell]_{10} \neq [j_k^*]_{10}$, there is at least one τ_h ($h \neq k$) which is not equal to α_h. Then, inputting $[\ell]_{10}$ to the cell will always turn off the jth gate. So, f_1 fault in jth gate can never be detected under this situation. This is contradiction.

The lemma is proved.

Theorem 1: In a single cell, as in Fig. 1b, any single s-a-f, except an f_1 fault in the OR gate and s-a-0 or s-a-1 on the cell output, can be detected by a unique detection condition.

Proof: The unique detection condition of f_1 fault in an AND gate is already proved in Lemma 4; what remains is f_1 on a control lead C_j or f_0 fault in jth AND gate.

a. To detect f_1 on C_j: C_j=0 and input $[j]_{10} = [\alpha_n \cdots \alpha_1]_2$ is the unique detection condition for this fault.

Sufficiency: Even if 0 is applied to C_j, f_1 on C_j will output 1 when input is $[j]_{10} = [\alpha_n \cdots \alpha_1]_2$. (Lemma 3) Error occurs.

Necessity: Since all other inputs will not sensitize the value of C_j to the putput and if C_j is not assigned opposite to 1, error can not be found at all.

b. To detect f_0 in jth AND gate: C_j=1 and input $[j]_{10} = [\alpha_n \cdots \alpha_1]_2$ is the unique detection condition for this fault.

Sufficiency: Even if 1 is applied to C_j, f_0 in jth AND gate will

cause the output to be 0 when input is $[j]_{10}=[\alpha_n \cdots \alpha_1]_2$. This is an error output by Lemma 3.

Necessity: Other inputs will not sensitize the putput of jth AND gate to the output, error can not be detected under those inputs. Even, if $C_j=0$ and if $[j]_{10} = [\alpha_n \cdots \alpha_1]_2$ is input, the output is still correct.

With the result of Lemma 4, the theorem is proved.

The result of the above lemma and theorem is illustrated briefly by the following example.

Example 2-9: Refer to Fig. 4.

1. input "a" s-a-0 can be detected if and only if $C_0=1$ and $x_2x_1 = 00$.
2. w_3 s-a-0 can be detected if and only if $C_0=1$, $C_1=0$, and $x_2x_1 = 01$. ($j = 0$, $k = 1$)
3. b s-a-1 can be detected if and only if $C_1=0$ and $x_2x_1 = 01$.
4. w_5 s-a-1 can be detected if and only if $C_2=1$, $C_0=0$ and $x_2x_1 = 00$. ($j = k = 2$)

From the above example, obviously, if a control pattern satisfies condition 2, it will be able to detect not only w_3 s-a-0 but also b s-a-1 if it exists. Therefore, some other patterns are necessary to distinguish between these faults in order to be able to locate faults. Fortunately, the all-0 pattern and well chosen (i,k)-patterns are found suitable for that job and will be discussed later.

According to the definition of j_k^*, the binary code representation of j and j_k^* are Hamming Distance(HD) ONE from each other on the kth

bit. From Lemma 1 and 2, for the jth gate in a cell with m primary inputs, an (i,k)-pattern with i+k=m and zero on its jth element will provide only one detection condition for an f_1 fault on the kth primary input to the j_k^*th gate, but an (i,k)'-pattern similarly characterized may provide any one of i+1 detection conditions for that kind of fault. Note that, in the above paragraph, we treat j and j_k^* inversely as stated in Lemma 4.

As a matter of fact, for a cell with m variables the index number of the (i,1)'-pattern where i+1=m is the maximum index number of any (j,k)'-pattern where j+k=m. Therefore, from the viewpoint of detection, the (i,1)'-pattern is the most powerful control pattern and has been used for multiple s-a-f's detection in the modular tree structure[1,2]. However, for the purposes of location, the (i,k)-pattern is most valuable because of its ability of partitioning the fault set.

The following algorithm can be used to detect multiple s-a-f's and locate a single s-a-f if only one s-a-f occurs in a single cell with m input variables.

2.4 Fault Location Algorithm I and Example

Fault Location Algorithm I

I. Apply all possible input combinations under the two (i,1)'-patterns where i+1 = m.

   (1) If no error is detected, terminate the procedure. (No s-a-f exists in that cell.)

   (2) If all outputs are constant 1 or 0, terminate the procedure. (f_1 fault in the OR gate or s-a-0 on the cell output.)

(3) If the detection test set for errors has two or more elements, terminate the procedure. (Multiple s-a-f's occur in that cell.)

(4) If the detection test set for errors has only one element and the error output is 0, terminate the procedure. (If the only element in the test set is w, let $[j]_{10} = [w]_2$, then the jth AND gate in the cell has an f_0 fault on it where $0 \leq j \leq 2^n-1$.)

(5) If the detection test set has only one element and the error output is 1, more tests are required to locate the fault.

II. Apply the all-0 pattern to the control leads of the cell and input w if w is the only element in the test set.

(1) If the output is 1, terminate the procedure. (Let $[j]_{10} = [w]_2$; the jth control lead of the cell is s-a-1.)

(2) If the output is 0, further tests are required.

Let k=1.

III. If the only element in the test set is w, let $[j]_{10} = [w]_2$. Apply the (m-k,k)-pattern whose jth element is 0 and input w to the cell.

(1) If the output is 1, terminate the procedure. (There is an f_1 fault on the kth primary input to the j_k^*th AND gate.)

(2) If the output is 0, set k = k+1, and repeat Step III.

Example 2-10: Refer to Fig. 4 and table I.

Those s-a-f's in the leftmost column of table I are chosen such that the algorithm is terminated at each different step.

In fact, those YES's, NO's, two-bit binary codes and blanks in

columns I(2), I(3), II and III provide enough clues for locating the s-a-f's in the corresponding leftmost columns. Now, the fault case in the sixth row, w_4 s-a-1, is discussed in detail.

After the detection test in step I, the NO in column I(2) negates any s-a-f on z or its equivalent. Then, "00" restricts those faults that might happen to: an f_0 fault in gate 0, s-a-1 on the control lead of gate 0, or an f_1 fault on the kth input wire to the 0_k^*th gate. Existence of the first two faults are negated by the two NO's in column I(4) and II. But, the YES in column III(1) verifies the f_1 fault on the 1st input wire to the 0_k^*th gate, that is, the x_1 input wire to gate 1. Under the single s-a-f assumption, the algorithm is terminated and w_4 s-a-1 is located to the wire level.

All the rest fault cases in this example can be located in the same way described above and are omitted here.

Note that since the two (i,1)'-patterns applied to the cell in Step I are complements of each other; the detection condition for each s-a-f which has a unique detection condition (Theorem 1) will be provided only once during Step I. In other words, only one error can appear throughout the whole detection test if only one s-a-f of that kind happens in the cell.

Table I: Demonstration of Fault Location Algorithm I(Refer to Figure 4.)

STEP:	I(2)	I(3)	I(4)	II	III	
	Constant output?	The error outputs occur for w=	Is the error output 0?	Is the output 1?	1 k=1 Is the output 1?	2 k=2 Is the output 1?
z s-a-1	Yes (1)					
w_5 s-a-0 and w_1 s-a-1	No	01 10				
w_1 s-a-1	No	01	Yes			
b s-a-1	No	01	No	Yes		
w_4 s-a-1	No	00	No	No	Yes	
w_5 s-a-1	No	00	No	No	No	Yes

3. FAULT LOCATION IN MODULAR TREES

In this chapter, fault location in combinational trees is discussed first; the result is applied to definite trees with slight modification[1,2]. Eventually, the number of tests required for fault location is tabled for both the combinational and definite tree.

3.1 Fault Location in Combinational Trees

The most important feature of a tree structure, the unique sensitized path for each control lead, exists not only in a single cell (see Lemma 3) but also in the whole tree[1,2]. That is, the value of any control lead in the tree can be propagated to the output only when a specific input combination is applied to the tree. Conversely, any input to a tree sensitizes a unique path from some control lead to the network output. Since, inside a tree, the output of a cell is a control lead of a cell in the next level, an error from the output of a faulty cell can be propagated to the output of the network only when the unique path is sensitized. The above characteristics of a tree simplifies the problem of fault location to a great extent.

3.1.1 Under Single s-a-f Assumption

Except for the last cell in the tree, faults on the output of a cell are indistinguishable from faults on the corresponding control lead of a cell at the next lower level. We treat them both as a s-a-f on the control lead and get the following conclusion.

For any single s-a-f in a cell at the highest level where its

control leads are accessible from the outside world, the input combination which may detect the fault and sensitize it is unique. This result follows the unique detection condition and the unique sensitizing path in a tree.

3.1.1.1 Awβ Form

To analyze faults buried within a tree, partition the tree into segments as shown in Fig. 6. The gate w is located in cell marked C where w represents the input in binary form that activates the gate. And, the corresponding control lead of gate w is wire w. Subtree T_C includes all cells between the output of the cell C and the root of the whole tree. Subtree T_w is the subtree whose root is wire w and whose leaves are also leaves of the whole tree. From previous discussion, we must input a fixed string (β) to T_C in order to sensitize the unique path from the output of cell C to the system output z. Input w is the input that activates gate w. Each input string (α) to subtree T_w propagates one of the two (i,1)'-patterns to cell C.

The notation Awβ will be used to represent the situation discussed above, where:

A is the set of all elements in I* having length ℓ for some integer ℓ. ($0 \leq \ell$)

$w \in I$

$\beta \in I^*$

and $\ell + lg(w) + lg(\beta) = n$ (lg(x) denotes the length of binary string x.)

If a single s-a-f exists anywhere in the tree, say in cell C shown

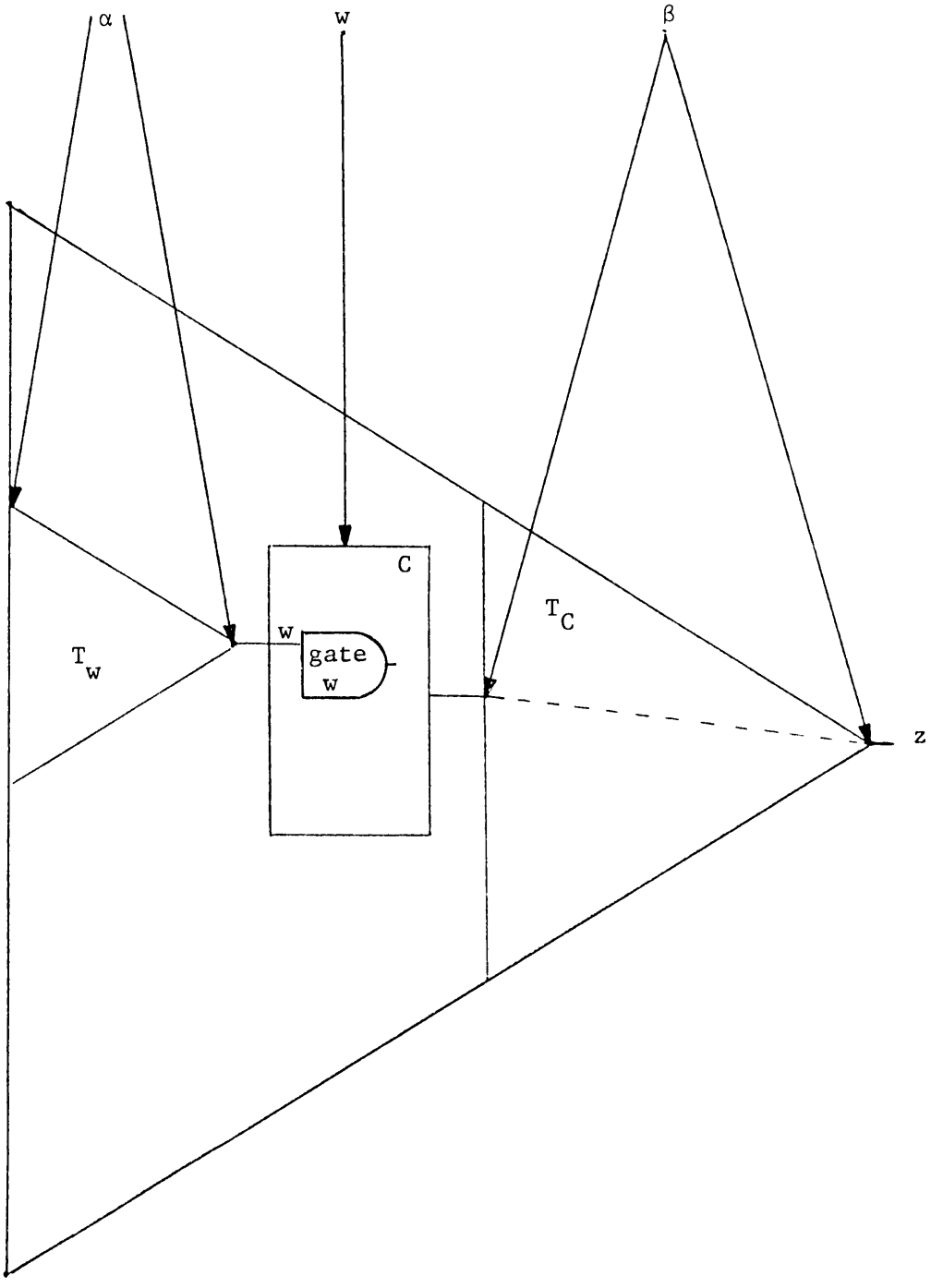

Figure 6: Modular Tree Segmented by αwβ where αεA

in Fig. 6, the set of input strings that produce erroneous outputs during the fault detection test to be described in Step I of Algorithm II will be in Awβ form for some integer ℓ. The set A consists of all input strings to T_w that propagate control patterns to faulty cell C capable of detecting the single s-a-f. Some input w with the appropriate pattern propagated by α provides the unique detection condition for that s-a-f. Then input string β is the unique input to T_C that propagates the faulty condition to the tree output.

Since β is unique for each faulty cell in the tree, the faulty cell can be located by determining β. Once the faulty cell is located by β, the remaining tests to locate the specific faulty wire will be identical to those in Algorithm I.

3.1.1.2 Fault Location Algorithm II and Example

Fault Location Algorithm II (Location of single s-a-f in a combinational tree with n primary input variables)

I. Apply all possible input combinations to the tree under the two (i,1)'-patterns where i+1 = n.

(1) If no error is detected, terminate the procedure. (No s-a-f exists in the tree.) If an error is detected, the set of inputs producing error outputs will be in Awβ form.

(2) If all error outputs are 0, terminate the procedure. (The faulty cell can be located directly by β. Let $[j]_{10} = [w]_2$, then the jth AND gate in the cell has an f_0 fault in it.)

(3) If any error output is 1, further tests are required.

II. Apply the all-0 pattern to the control leads of the tree and input

$\alpha w\beta$ (any $\alpha \varepsilon A$). If the output is 1, terminate the procedure. (Let $[j]_{10} = [w]_2$, the jth control lead of the faulty cell which is located by β is stuck at 1.)

Let k = 1.

III. Apply the (m_i-k,k)-pattern, where $m_i = lg(w)$, to the faulty cell that will propagate 0 to the jth control lead. (That is, to the whole tree, a $(n-k-lg(\alpha),k+lg(\alpha))$-pattern which will propagate a 0 to the jth control lead of the faulty cell under any $\alpha(\alpha \varepsilon A)$) and input $[w]_2$.

(1) If the output is 1, terminate the procedure. (An f_1 fault occurs on the kth primary input to the j_k^*th AND gate in the faulty cell located by β.)

(2) If the output is 0, set k=k+1, repeat Step III.

We now illustrate the above argument by the following example.

Example 10: Refer to Figure 7, table II and III.

Table II displays the β that uniquely corresponds to each cell. Table III contains an analysis of five faults. It should be noted that $w = \beta = \lambda$(null string) (see the bottom row in table III), indicates s-a-f on the tree output. All the rest of the table is self-explanatory.

3.1.2 Under Multiple s-a-f's Assumption

Unfortunately, the error detecting inputs found at STEP I, Algorithm II, may be in $Aw\beta$ form for some multiple s-a-f's as well as for all single s-a-f's. In this section, a study of multiple s-a-f's will be undertaken.

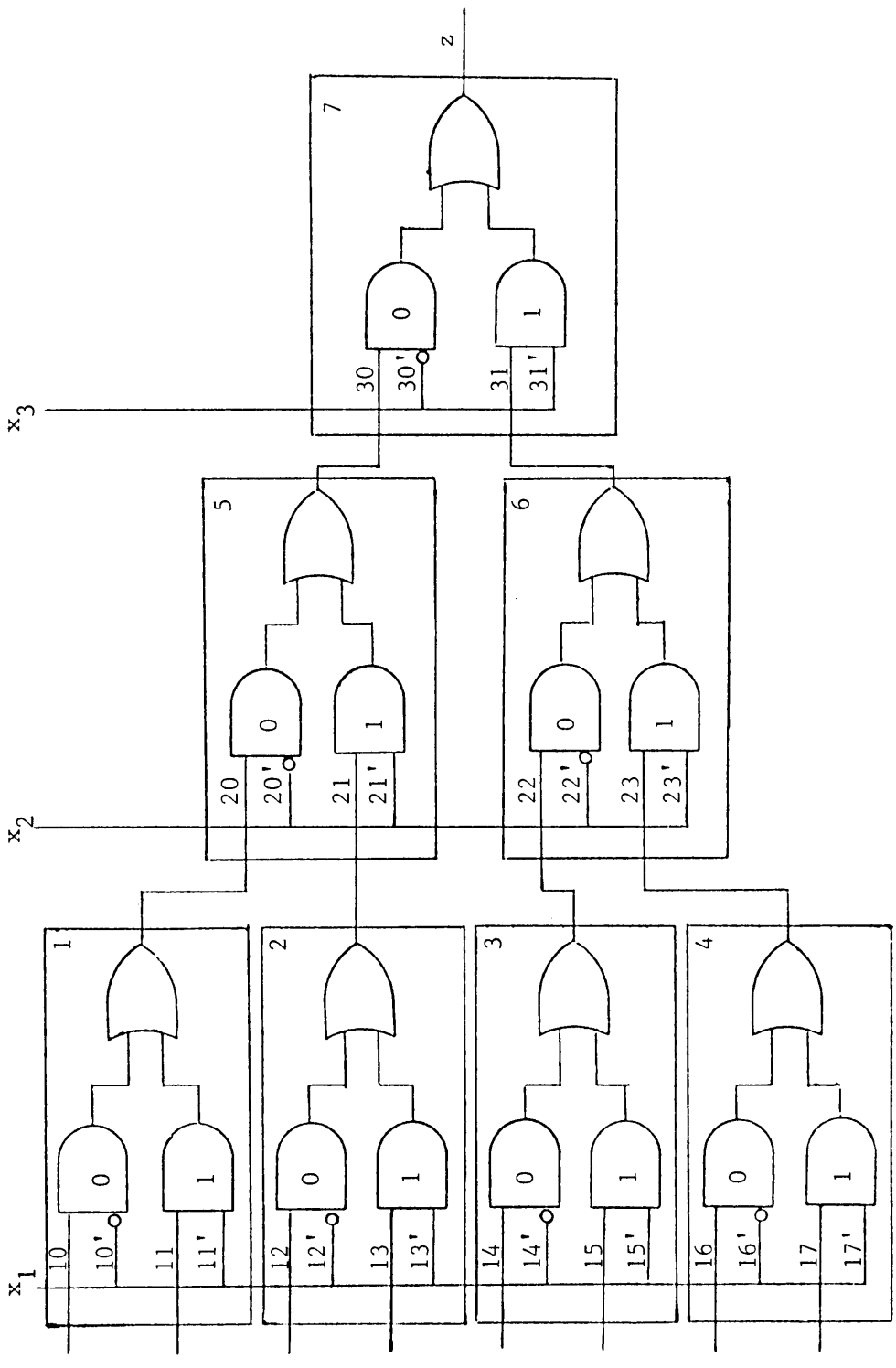

Figure 7: Binary Modular Tree with Three Input Variables x_1, x_2 and x_3

Table II: Displaying of the Unique β of Each Cell in Figure 7

Cell	Unique β ($x_1x_2x_3$)
1	-00
2	-10
3	-01
4	-11
5	--0
6	--1
7	---

Table III: Demonstration of Fault Location Algorithm II(Refer to Figure 7.)

	Algorithm II Step:	I		II	III	
Fault	(1) List all input combinations with error output in Awβ form. If none, halt.	(2) Is any error output 0?	(3) Is any error output 1?	Is the output 1?	k=1 Is the output 1?	Fault Diagnosis
13 s-a-0	$x_1x_2x_3$=110 A=ϕ, w=1, β=10	Yes				2nd cell, f_0 on 1st AND gate (s-a-0)
22 s-a-0 (or) 22' s-a-1	$x_1x_2x_3$=001,101 A={0,1}, w=0 β=1	Yes				6th cell, f_0 on 0th AND gate (s-a-0)
23 s-a-1	$x_1x_2x_3$=011,111 A={0,1}, w=1 β=1	No	Yes	Yes		6th cell, 1st control lead s-a-1
30' s-a-0	$x_1x_2x_3$=101,011 001,111 A={00,01,10,11}, w=1, β=λ (λ:null string)	No	Yes	No	Yes	7th cell, f_1 fault on 1st primary on 0th AND gate
z s-a-1	$x_1x_2x_3$=000,001, 010,011, 100,101, 110,111 A={000,001,010, 011,100,101, 110,111} w=β=λ (λ:null string)	No	Yes			s-a-1 on the output z or its equivalent

3.1.2.1 Pseudo-Single s-a-f

Definition:

A pseudo-single s-a-f is a multiple s-a-f that is indistinguishable from a single s-a-f during the detection test in STEP I, Algorithm II.

A pseudo-single s-a-f is located to within a specific area of the tree by Lemma 5 and Theorem 2. More tests are then applied to locate the faults to the wire level.

Lemma: During the multiple s-a-f's detection test for a single cell, the only distinguishable multiple s-a-f's in that cell that can simulate in the same cell a single s-a-f (which has a unique detection condition) must look like an f_1 fault. These can be distinguished by applying (i,k)-patterns to the cell. (i+k=m and $k \geq 1$)

Proof: Since those single s-a-f's in a cell mentioned here are those which have a unique detection condition (Theorem 1). By the remark at the end of Chapter 2, only one error can be detected by the multiple s-a-f's detection test if a single such s-a-f occurs.

f_0 fault: If the only error output is 0 and is detected by the test with C_j=1 and input $[j]_{10}$, then a single f_0 fault in the jth AND gate is simulated.

Assume that no f_0 fault exists in the jth AND gate, then this gate will be activated and output 1 when input is $[j]_{10}$ and C_j=1. But, the actual output of the cell is 0 now, that means, the output 1 from the jth AND gate is blocked somewhere between the gate output and the cell output by some s-a-f. The fault has to be a s-a-0 on the cell output, but by Theorem 1, this s-a-f will be detected by some other tests

for sure. This is a contradiction.

f_1 fault: Assume the only error output is 1 and is detected by the test with C_j=0 and input $[j]_{10}$ and that C_j is not stuck at 1 and no f_1 faults occur in a j_k^*th AND gate for any k which are supposed to be detected by this test.(see Example 2-9)

Assume the error is caused by s-a-f's in the ith AND gate where $0 \leq i \leq 2^m-1$ and $i \neq j$.

Case 1: C_i is assigned to be 1 by the (i,1)'-pattern which assigns C_j=0.

C_j can not have s-a-0 on it since the error output in question is 1. Therefore, there are at least two f_1 faults in the gate and suppose one of them is on the input wire of the hth input variable to the ith gate. Since C_i is set 1, so, $C_{i_h^*}$ will be certainly assigned to be 0 by that (i,1)'-pattern(lemma 2). That means when the input to the cell is $[i_h^*]_{10}$, the correct output should be 0. But, f_1 faults in the ith AND gate will cause the gate to be activated by input $[i_h^*]_{10}$ and the cell output will be 1 since C_i=1. This is a contradiction.

Case 2: C_i is assigned to be 0 by the (i,1)'-pattern which assigns C_j=0.

In this case, not only does some f_1 faults in the ith AND gate exist but also the ith control lead of the cell has to be stuck at 1. These f_1 faults however guarantee another occurrence of error output 1 under the same pattern and input $[i]_{10}$. This is a contradiction.

The lemma is proved

Theorem 2: In a tree, a multiple s-a-f that simulates a single s-a-f in some AND gate j under multiple s-a-f's detection test must exist either

(1) in the subtree with control lead C_j as the root and simulate a s-a-0 or a s-a-1 on the control lead C_j.

or (2) in the same cell where gate j is located and can be distinguished by applying (i,k)-patterns where $1<k$.

Proof: Assume the set of inputs which propagate error outputs 1(or 0) to the tree output under the multiple s-a-f's detection test is in the Awβ form, that means either a real or a pseudo single s-a-f happens in the tree. Then, T_w, C, and T_C can be located as shown in Fig. 6. Now, let's segment the tree into two regions as shown in Fig. 8. Region II is the subtree with the cell C as the output cell; the rest of the tree is region I.

First, we want to prove that no s-a-f may exist in Region I, if the set of the tests with error outputs is in Awβ form.

(I) Assume there are s-a-f's scattered around Region I.

Let C' be the faulty cell in Region I nearest to the output of the tree .

Case IA: Cell C' is not on a path from cell C to the tree output.

As shown in Fig. 9, the subtree T'_w is rooted at C' and its output can be propagated to the tree output only when β' is input to the subtree T'_C

That means only β' can sensitize the error caused by a s-a-f in cell C' to the tree output. Certainly, β' is different from β.

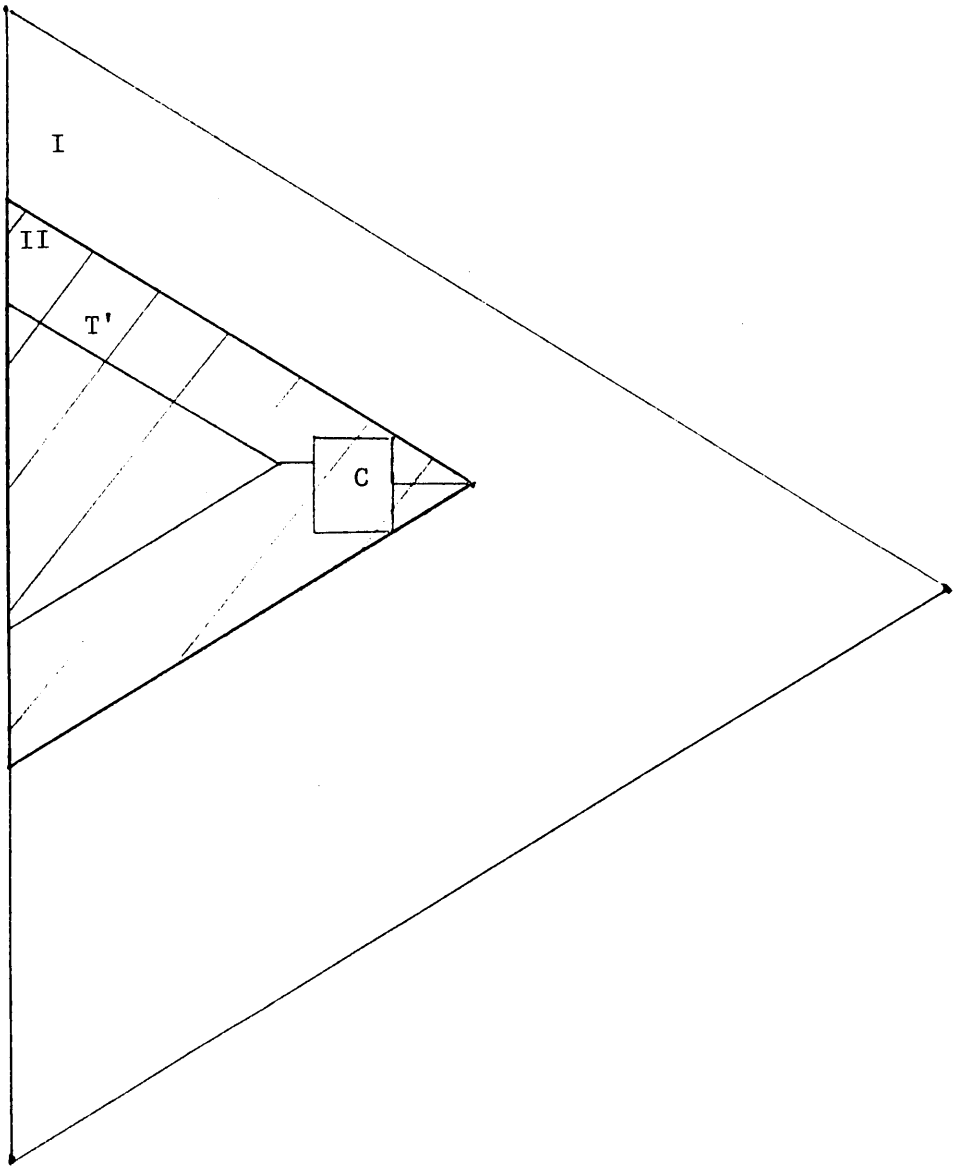

Figure 8: Modular Tree Segmented into Two Regions
Region II is the Subtree Rooted at the Output of Cell C. The Rest of the Tree forms the Region I.

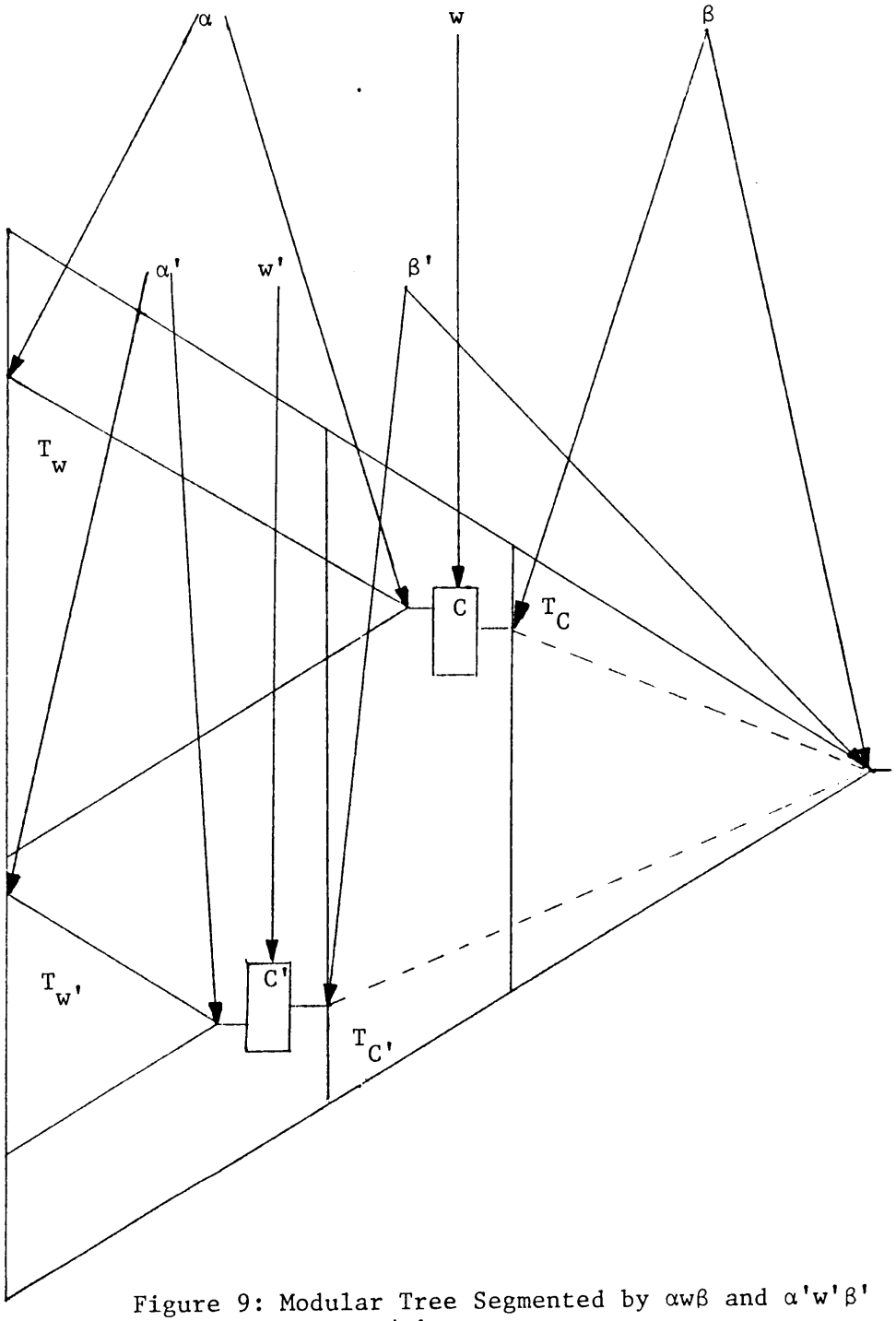

Figure 9: Modular Tree Segmented by αwβ and α'w'β' where β≠β'

Thus, the error detecting test set is not in Awβ form and it is a contradiction.

Case IB: Cell C' is on a path from cell C to the tree output.

The tree may be partitioned into segments as in Fig. 10 where β is divided into three strings α', w' and β'. But, this time, α' is unique.

IB1. If some s-a-f in C' has input $[w'']_2$ as part of its unique detection condition which has to be provided for detecting it, but $[w']_2 \neq [w'']_2$. Now let $[w']_2=[j']_{10}$, and $[w'']_2=[j'']_{10}$.

Unless these s-a-f's are masked by some other s-a-f's which is discussed in IB2 or there are contradictions.

IB2. If s-a-f's in C' have $[w']_2$ as part of their detection conditions then they have to fall into one of the following three categories of s-a-f's.(see Theorem 2)

a. f_1 on input wire of kth primary input to the j'^{*}_{k}th AND gate of C' where $1 \leq k \leq m_{C'}$, and $m_{C'}= \lg(w')$.

b. f_1 on the j'th control lead of C'

c. f_0 in the j'th AND gate of C'

Faults of type a. and b. may occur simultaneously, but faults of type a will be masked by those of type b. Faults of type c will mask those of type b but not of type a. So, faults of type a and c can't occur simultaneously to simulate a s-a-f.

Now, we consider them separately.

a. Detection conditions for these faults are $C_{j'}=0$, $C_{j'^{*}_{k}}=1$ and input = $[w']_2$.

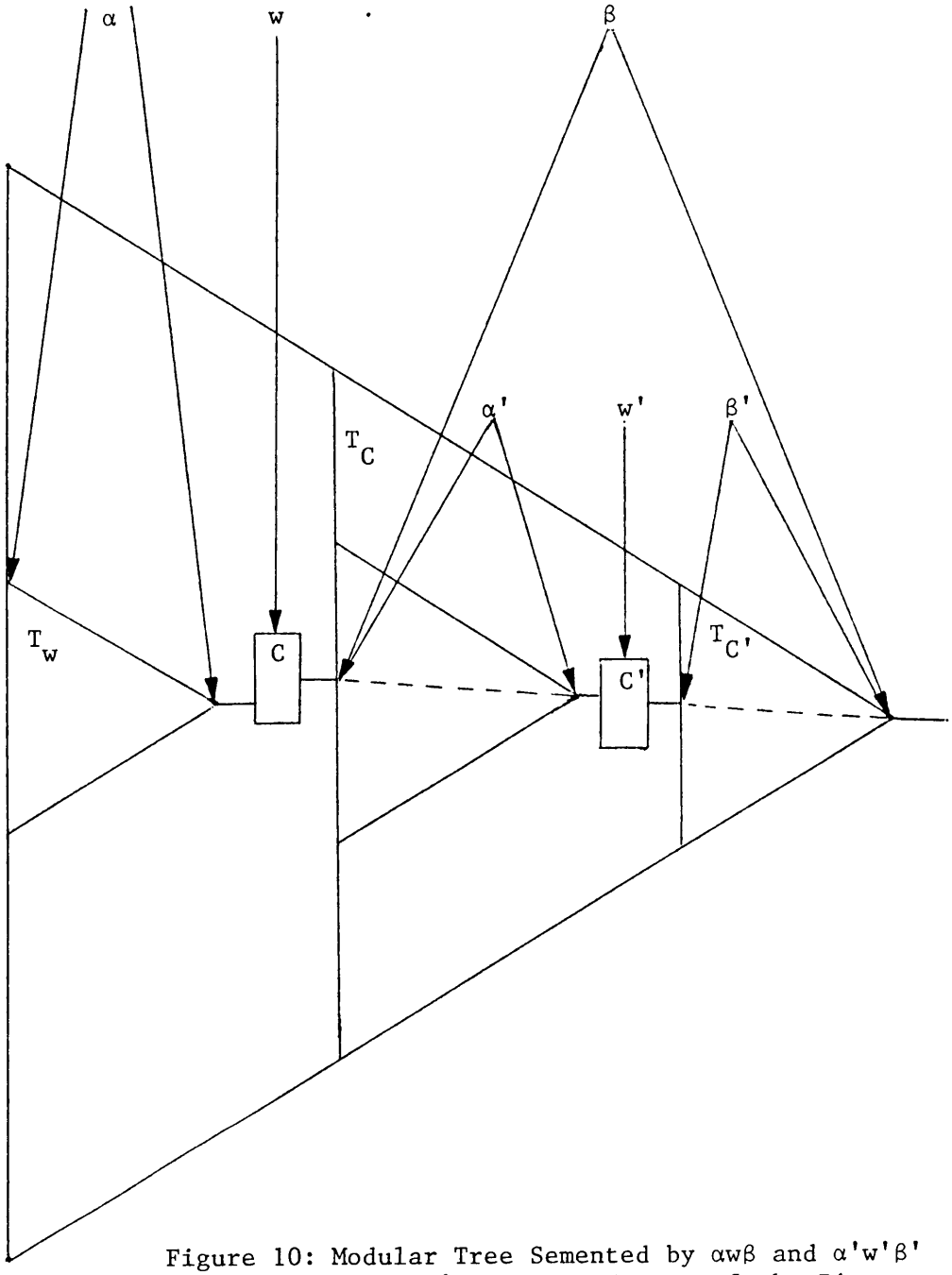

Figure 10: Modular Tree Semented by αwβ and α'w'β' where β' is the End Part of the Binary String β.

There is at least one input string $\tau \neq \alpha w \alpha'$ and $lg(\tau) = lg(\alpha w \alpha')$ where $\alpha \in A$ which is supposed to propagate the desired $(i,1)'$-pattern to C' to detect these f_1 faults.

(1) If the pattern is propagated to C' correctly at $C_{j'}$, and $C_{j'^*_k}$, that is, $C_{j'}=0$ and $C_{j'^*_k}=1$.

If there are one or more f_1 faults in gate j'^*_k of C' other than the $C_{j'^*_k}$ s-a-1, then, certainly, cell C' will output 1 under input $[\tau w' \beta']_2$ but $\tau w' \beta' \notin Aw\beta$. It is a contradiction.

(2) If the pattern propagated to C' sets $C_{j'}=0$ and $C_{j'^*_k}=0$, then f_1 faults in gate j'^*_k will not be detected by inputting $[\tau w' \beta']_2$. However, this error will be either detected by inputting $[\tau w'^*_k \beta']_2$ (Contradiction occurs.) or masked by some gate,say v, which outputs 1 at that time.($[w'^*_k]_2 = [j'^*_k]_{10}$)

(2a) If $[v]_{10} = [(j'^*_k)^*_{k'}]_{10}$ for some $k' \neq k$, then C_v should have 0 on it. But, it outputs 1 as mentioned, therefore, either C_v is stuck at 1 or some erroneous propagation happens again. Anyhow, this fault will be detected under input $[v]_{10}$. This is a contradiction.

(2b) If $[v]_{10} \neq [(j'^*_k)^*_{k'}]_{10}$ for any k',and if $C_v=1$ is propagated wrong, error will be output under input $[v]_{10}$. This is a contradiction.

But, even if $C_v=1$ is correctly propagated, error still occurs under input $[v^*_k]_{10}$ for some k. This is a contradiction.

(3) Assume the pattern propagated to C' set $C_{j'}=1$.

(3a) If there is no f_0 fault in gate w', then error certainly occurs when input is $[\tau w'\beta']_2$. This is a contradiction.

(3b) If there are f_0 faults in gate w', they will be delt in the discussion in case c.

b. Since this fault can not be masked under multiple s-a-f's detection test, therefore, the set of inputs which output errors at least contains a subset in A'w'β' form where $lg(\tau')=lg(\alpha)+lg(w)+lg(\alpha')$ and $\tau'\varepsilon A'$. This is a contradiction.

c. This fault can be detected when $C_j=1$ and by inputting $[w']_2$.

There is at least one input string $\tau'=\alpha w\alpha'$ where $\alpha\varepsilon A$ and $lg(\tau')= lg(\alpha w\alpha')$ which is supposed to propagate the desired (i,1)'-pattern to C' to detect this f_0 fault in gate w'.

So, the error caused by f_0 fault in gate w' should be sensitized by inputting $[\tau'w'\beta']_2$ unless it is masked. That means some other AND gate, say gate v', outputs 1 when the input is $[\tau'w'\beta']_2$. Then the argument is the same as those in (2a)and (2b) except for v now is v' and $(j'^{*}_{k})^{*}_{k'}$ is j'^{*}_{k}. Similarly, all the arguments end in contradictions.

Now, we conclude that there are no s-a-f's which may exist in Region I with the error detecting test set in Awβ form. Therefore, s-a-f's in the tree must exist in Region II as shown in Fig. 8.

(II) 1. Multiple s-a-f's except those which can be distinguished under

(i,k)-patterns ($1 \leq k$) can not simulate a single s-a-f in cell C by Lemma 5.

2. Any s-a-f in Region I but not in C and T_w will produce an error to some control lead of C under some input α'' to that part. Since, during the detection test, all possible input combinations are applied to the tree under the two (i,1)'-patterns, all possible w's will appear between α'' and β. That means, a thorough multiple s-a-f's detection is undertaken there. In the proof of Lemma 5, this is exactly the case which has a s-a-f on some control lead and it may not simulate another s-a-f. So, once these s-a-f's occur, $Aw\beta$ can never happen, so it is a contradiction.

So, the area where the multiple s-a-f may exist to simulate a single s-a-f is already confined to either the cell C or the subtree T_w.

(III) 1. If the multiple s-a-f exists in cell C, by Lemma 5, they must look like an f_1 fault in cell C, and can be distinguished by applying (i,k)-patterns and inputting $[j]_{10}$ to the cell where $1 \leq k$.

2. If the multiple s-a-f exists in subtree T_w, since the subtree is rooted at C_j, obviously, C_j s-a-0 or s-a-1 is the only fault which it may simulate and also make the set of inputs which output errors in the $Aw\beta$ form. The theorem is proved.

So, if the result of the multiple s-a-f's detection test indicates the existence of a single s-a-f (real or pseudo) inside a tree, by the above theorem, we know that the fault exists in the cell C

located by β, or in the subtree T_w where $[w]_2=[j]_{10}$.

If the fault is an f_0 in the jth gate of cell C or an f_1 on the jth control lead of cell C, then no matter which other s-a-f's may exist in T_w or f_1's in gate j_k^* of cell C, possible utilization of the faulty tree is uniquely defined. That is, the function which can be correctly performed by the faulty tree must have on all-0 (or all-1) pattern on the leaves of T_w.

However, if in cell C, some f_1 faults on the kth input variable to the j_k^*th gate of cell C occur, but niether an f_1 nor an f_0 fault occurs on C_j, the situation is quite different and further testing is required.

First, if f_1's occur in the j_k^*th gate, all error outputs during the detection test would be 1 and the multiple fault will simulate an f_1 fault on C_j of cell C. That is, the input is $[j]_{10}$ and $C_{j_k^*}=1$ where C_j should be 0 by the proof of Lemma 2. But, in this case, whether $C_j=0$ or $C_j=1$, the error output will always be 1. In other words, the output of T_w has no effect on the tree output when those f_1 faults occur in cell C. Unfortunately, the exact f_1's that occur in T_w will effect tree utilization. That means, we have to worry about locating them even after s-a-f's in cell C have been located. A recursive procedure for fault location is inevitable in this situation. Note that there can be no detectable f_0 in T_w in this situation, because that would imply an error output of 0 during the detection test.

3.1.2.2 Fault Location Algorithm III and Example

Fault Location Algorithm III (For detecting multiple s-a-f's and locating real-single or pseudo-single s-a-f's in a combinational tree with n input variables.)

I. Apply all possible input combinations to the tree under the two (i,1)'-patterns where i+1=n

(1) If no error is detected, terminate the procedure. (No s-a-f exists in the tree.)

(2) If errors are detected, but the set of inputs producing error outputs is not in Awβ form, terminate the procedure. (Non-locatable multiple s-a-f's occur.)

If errors are detected, and the set of inputs producing error outputs is in Awβ form, then continue.

(3) If all error outputs are not the same, terminate the procedure. (Non-locatable multiple s-a-f's occur.)

(4) If all error outputs are 0, terminate the procedure. (The faulty cell can be located by β. Let $[j]_{10} = [w]_2$; the jth gate in the cell has an f_0 fault on it.)

(5) If all error outputs are 1, and if w=β=λ(null string), terminate the procedure (s-a-1 on output of the tree occurs.); otherwise further tests are required.

Let W=w, B=β, J=j, and A'=A.

II. Locate the faulty cell C by B, then the subtree having the Jth

control lead of the faulty cell as its root is named as T_w. Apply one of the two (i,1)'-patterns to T_W while setting all the remaining control leads of the tree to 0. The set of test inputs is { $\gamma WB \mid [\gamma]_2 = [h]_{10}$ and $C_h = 0$ in the (i,1)'-pattern applied to the subtree T_w }. Apply the other (i,1)'-pattern to T_W and repeat the above procedure under that pattern.

(1) If no error is detected, go to III. (No s-a-f exists in T_W.)

(2) If all outputs are 1, terminate the procedure. (A s-a-1 on C_J of the faulty cell or its equivalent occurs.)

(3) If errors are detected, but the set of inputs producing error outputs is not in $Aw\beta$ form, then terminate the procedure. (Non-locatable multiple s-a-f's exist in T_W.)

(4) If the set of input detecting error outputs is in $Aw\beta$ form where $\beta=\beta'WB$ for some $\beta'\varepsilon I$. Go to III'.

Let K=1 and $\alpha'\varepsilon A$

III. (1) Apply to T_J one of the two (n-lg(α')-K, lg(α')+K)-patterns which, under fault-free conditions and any input, always propagates an (m_i-K,K)-pattern whose Jth element is 0 to cell C. Apply 0 to all the remaining control leads of the tree. Input to the tree any γWB which gives the correct output in Step II. (If output is 1, there is an f_1 fault on the Kth primary input to the J_K^*th AND gate of the cell C.)

(2) Let K=K+1. If $m_i-K<0$ (m_i=lg(W)), then, terminate the procedure; otherwise, go to III(1).

III' (1) The same as Step III(1)

(2) Let K=K+1. If m_i-K$\geq$0 (where m_i=lg(W)) then go to III(1)'; otherwise, let A'=A, $[J]_{10}=[w]_2$, W=w, and B=βWB. If A'=φ, go to II. Otherwise, still go to II, but apply the all-0 pattern to the tree instead of the (i,1)'-patterns.

Example: Refer to Fig. 7 and Table IV.

This is example is similar to the previous two examples demonstrating the functions of Algorithm I and II. The primary difference in this case occurs for s-a-f's in the last row of the leftmost column where the recursive part in the algorithm is illustrated.

Cell 7 where wire 31' s-a-1 exists is located by β=λ (given in column I(2)). Wire 30 and the subtree rooted to it are tested by Step II for f_1 faults. The test result indicates the existence of a single f_1 fault(real or pseudo), but wire 30 stuck at 1 is eliminated by "NO" in column II(2). Then, wire 31' s-a-1 is locatable and located by the positive answer to Step III'(1), w=0 and β=λ. Since the existence of an f_1 fault in the subtree rooted at wire 30 has been verified, the algorithm is applied to the subtree again starting at Step II. Eventually, wire 21' s-a-1 is located by the "YES" in column III'(1), w'=0 and β'=0; and wire 10' s-a-0 is located by the "YES" in column III(1), w"=1 and β"=00.

3.2 Fault Location in Definite Trees

Fig. 11a and 11b show a k_d-ary definite tree of order of definiteness(depth) d and the cell structure, respectively.

Table IV: Demonstration of Fault Location Algorithm III(Refer to Figure 7.)

	I					II			III(1)	III'(1)
	Inputs which produce error outputs	(2) In form $Aw\beta$?	(3) Are all error outputs a constant ?	(4) Are all error outputs 0?	(5) Are all error outputs 1?	Inputs which produce error outputs	(2) Are all outputs 1?	(3) & (4) In $Aw\beta$?	K=1 Is output 1?	K=1 Is output 1?
20 s-a-1 15 s-a-0	000 100 101	No								
14 s-a-0 15 s-a-0 (equivalent to 22 s-a-0)	001 101	A={0, 1} w=0 β=1	Yes	Yes						
20 s-a-1	000 100	A={0, 1} w=0 β=0	Yes	No	Yes	000 100			Yes	
20' s-a-0	110 010	A={0, 1} w=1 β=0	Yes	No	Yes	None				Yes
31' s-a-1 21' s-a-1 13' s-a-0	000 110 010 100	A={00, 01, 10, 11} w=0 β=λ	No							
31' s-a-1 21' s-a-1 10' s-a-0	000 110 010 100	A={00, 01, 10, 11} w=0 β=λ	Yes	No	Yes	000 100	No	A'={0, 1} w'=0 β'=0		Yes
						100	No	A''=ϕ w''=1 β''=00		Yes
							No		Yes	

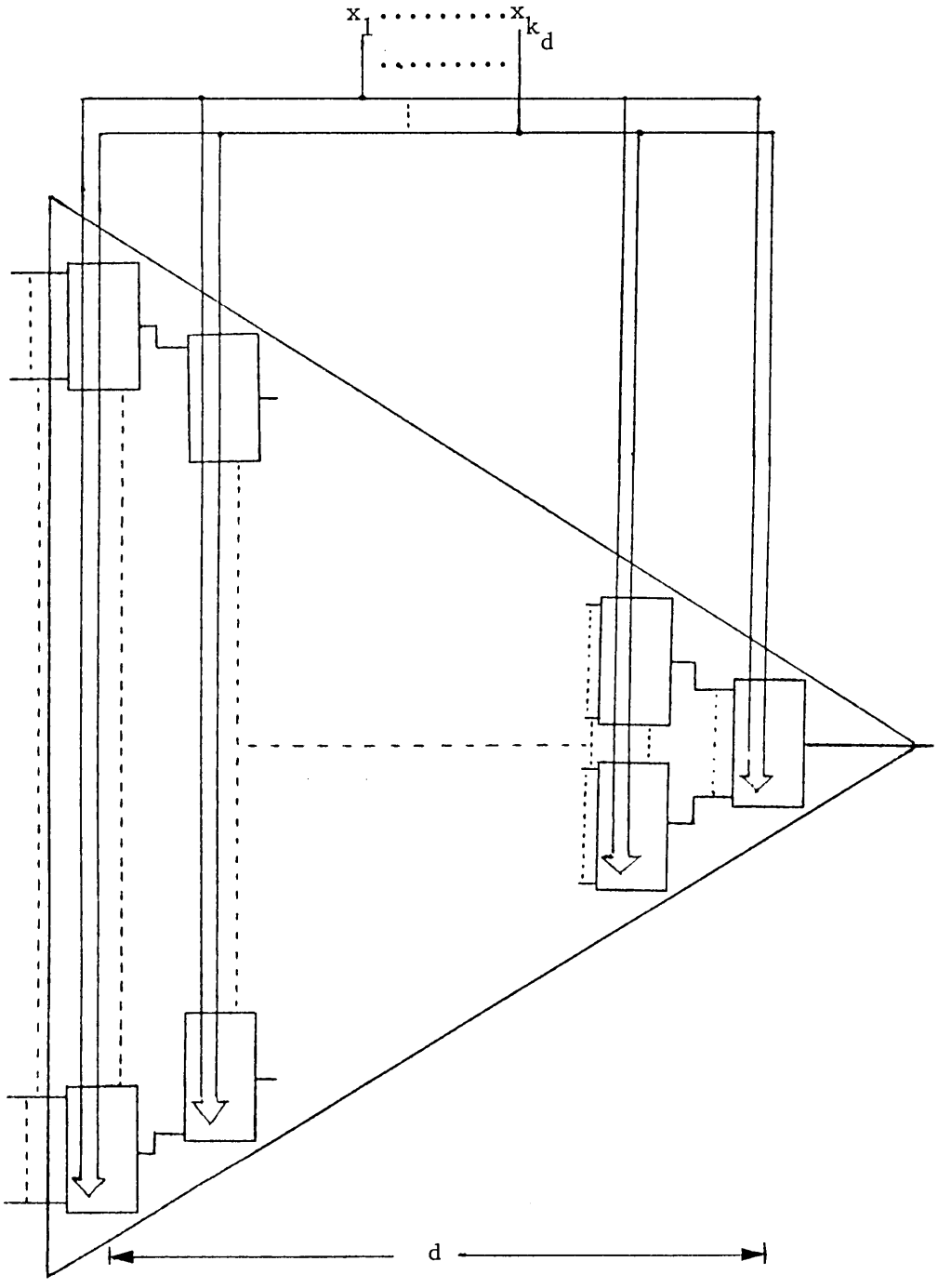

Figure 11a: Modular Tree used to Realize k_d-ary Definite Machines with Definiteness d

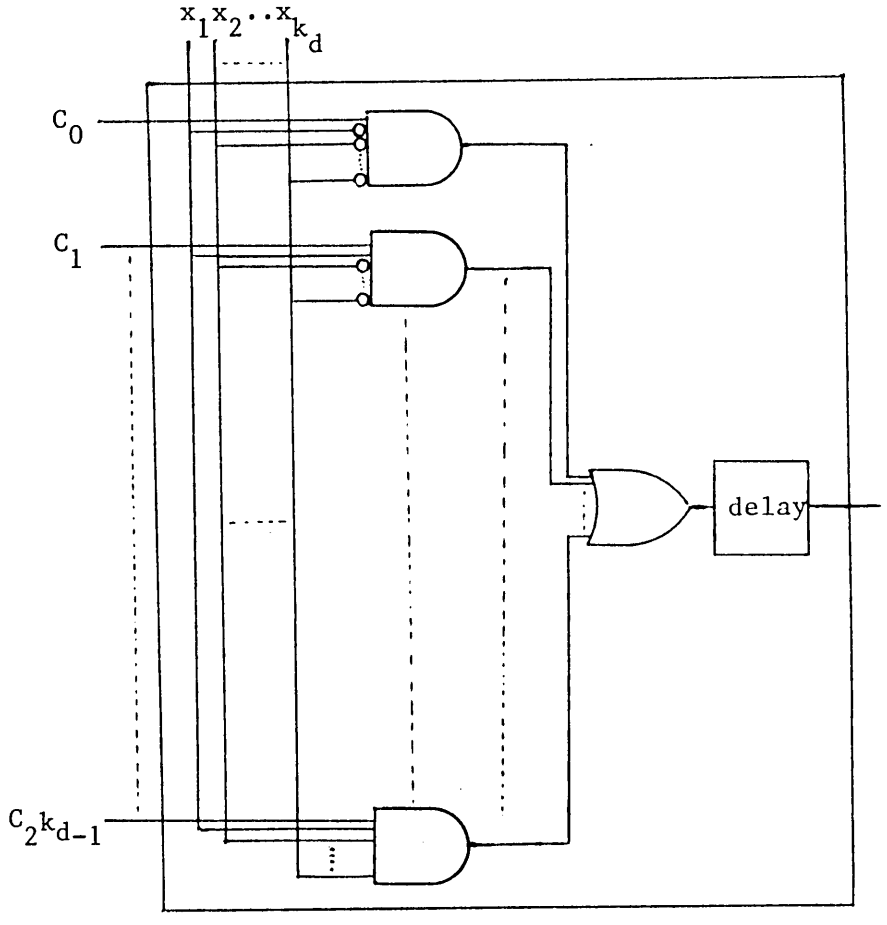

Figure 11b: Structure of Cell used in Figure 11a

All of the above discussion on combinational trees is also relevant to definite trees [6,7,8]. However, each output of a definite tree is dependent on the inputs at the previous d time-intervals. In a definite tree, a time input sequence is used for each input combination to a similar combinational tree. So, the number of tests for detecting and locating s-a-f's will increase. But, by efficiently arranging the input sequences, overlapping among them may reduce the number of tests considerably [4,12].

Since, the algorithms for combinational trees can be applied directly to the definite tree except for changing all the input combinations used in the algorithms to time input sequences, detailed algorithms for fault detection or location in definite tree are omitted. However, the number of tests for those purposes are formulated in the next section.

3.3 Upper Bounds on the Length of the Fault Location Experiment

The upper bound on the number of tests for locating a (real or pseudo) single s-a-f in combinational trees and definite trees are formulated and listed in Table V.

The underlined part in those expressions in Table V gives the magnitude of multiple s-a-f's detection test(N_D); the rest gives the number of tests for locating single (real or pseudo) s-a-f's(N_L).

The following example describes the effects of variation on m, m_h, k_d and d on the number of tests for locating s-a-f's in combinational and definite trees.

Example: Refer to Fig. 1a and 11a and Table VI, VII, and VIII.

All the combinational trees discussed in this example are assumed uniformly constructed. That is, m_h is the same integer for all h. ($1 \leq h \leq k$)

Table VI indicates the variation of the number of tests with m for detecting s-a-f in combinational tree under the single s-a-f assumption. The worest faulty condition that a combinational tree may have is that multiple locatable s-a-f's exist in each level of that tree. The numbers of tests needed when m_h of a tree varies are given in Table VII, with the assumption that the worst case occurs in the tree. Table VIII displays the growth of the number of tests with the increase of d where $k_d \cdot d$ is a constant

Table V

Refer to Figure 1a.

m_h : the number of input variables to the h-th level of a combinational tree

m : the maximum m_h

n : the total number of input variables to the combinational tree ($n = \sum_{h=1}^{k} m_h$)

Refer to Figure 11a.

d : the depth of a definite tree

k_d : the number of input variables to the definite tree

tree type \ fault type	under single s-a-f assumption	under multiple s-a-f assumption
combinational	$2\cdot(2^n)+m+1$	$2\cdot(2^n)+\sum_{h=2}^{k}(2^{n-\sum_{i=2}^{h} m_i}+m_h)+m_1+1$
definite	$2\cdot(2^{k_d\cdot d}+d-1)+d\cdot(k_d+1)$	$2\cdot(2^{k_d\cdot d}+d-1)+(d-1)\cdot(2^{k_d\cdot d}+d-1)+(d+1)\cdot d\cdot k_d+d$

Table VI

	Single s-a-f assumption					
	m = 1	2	3	4	5	6
no. of detection tests N_D	128	128	128	128	128	128
no. of location tests N_L	2	3	4	5	6	7

Table VII

	Under Multiple s-a-f's Assumption			
	m_h=1	2	3	4
no. of detection tests N_D	128	128	128	128
no. of location tests N_L	67	27	15	7

Table VIII

	Single s-a-f assumption			
d=	1	2	3	6
no. of detection tests N_D	128	130	132	138
no. of location tests N_L	7	8	9	12

4. DETECTION OF BASC AND SHORT CIRCUIT FAULTS IN MODULAR TREES

In this chapter, the problem of detecting BASC faults and short circuit faults is discussed under certain restrictions. The fault may involve only two wires and these two wires can be only primary inputs, primary input wires or control leads to a single cell. For instance, in Fig. 13, x_1 and x_2 are primary inputs, w_{12} and w_{32} are primary input wires, and C_0 and C_2 are control leads. From now on, R fault will be used to represent the BASC and SC faults in the restricted sense described above.

For the convenience of later discussion, R faults which may happen in a single cell are classified into 9 froups, and short symbolized notation is given to each group respectively.

Group 1: Control lead C_i is short circuited to control lead C_j. (C_i SC C_j)

Group 2: Control lead C_i is broken and short circuited to control lead C_j. (C_i BASC C_j)

Group 3: Control lead C_i is short circuited to a primary input x_k. (C_i SC x_k)

Group 4: Control lead C_i is broken and short circuited to a primary input x_k. (C_i BASC x_k)

Group 5: Primary input wire w_{ik} is broken and short circuited to control lead C_j. (w_{ik} BASC C_j)

Group 6: Primary input wire w_{ik} is broken and short circuited to one of the other primary inputs, say x_h. (w_{ik} BASC x_h)

Group 7 Primary input x_k is short circuited to another primary input x_h. $(x_k \text{ SC } x_h)$

Group 8 Primary input x_k is broken and short circuited to another primary input x_h. $(x_k \text{ BASC } x_h)$

Group 9 Primary input x_k is broken and short circuited to control lead C_i. $(x_k \text{ BASC } C_i)$

Notice that none of these R faults consists of two broken wires, because, in that case, they both would turn out to be s-a-f's.

4.1 Detection Condition for an R Fault in a Cell

The detection condition for an R fault is discussed in the following paragraph, and test sets for detecting single R-faults in a cell or in modular trees are designed.

Before a detection test can be designed, conditions which has to be provided for detecting an R fault should be studied. Except for groups 1, 2, 7 and 8, each group requires one or two different detection conditions. The detection conditions with corresponding faults are enlisted below, and short proofs are given.

Notation: Refer to Fig. 12

$[\ell]_{10} = [i_k^*]_{10}$ where $1 \leq k \leq m$ will be used throughout this chapter.

w_{ik}: the input wire from input variable x_k to gate i

C_i : the control lead of gate i

x_k : the common wire connecting input variable x_k to every AND gate in the cell

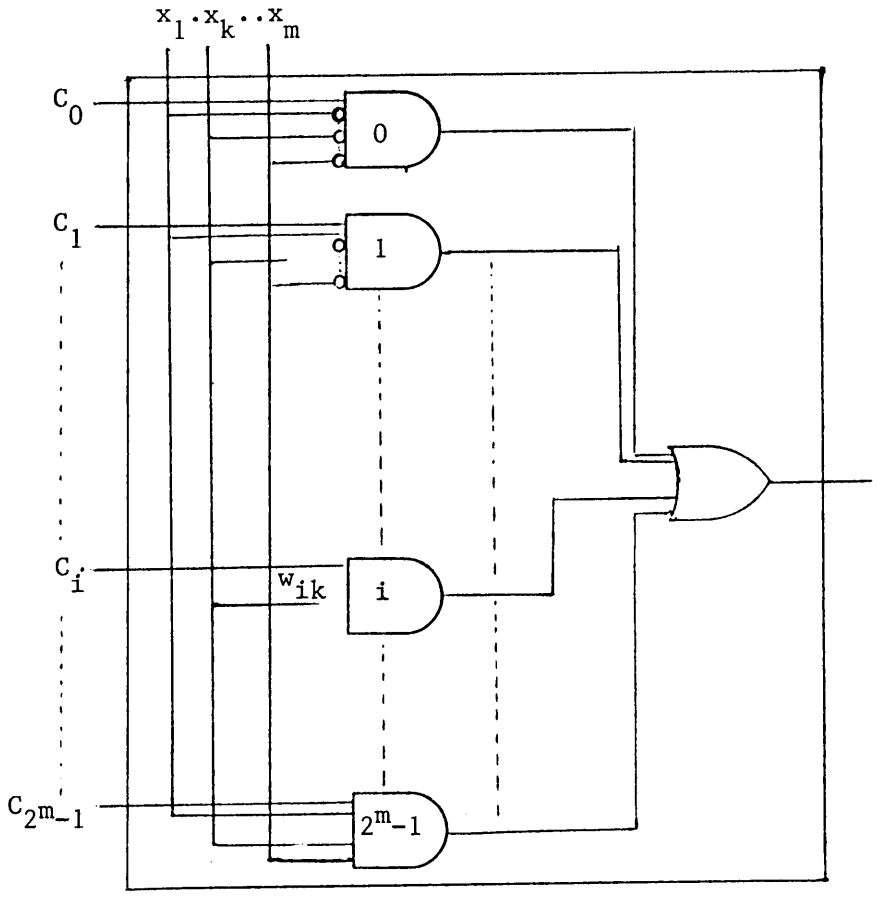

Figure 12: Cell with m Input Variables for Displaying Notations C_i, w_{ik}, and x_k

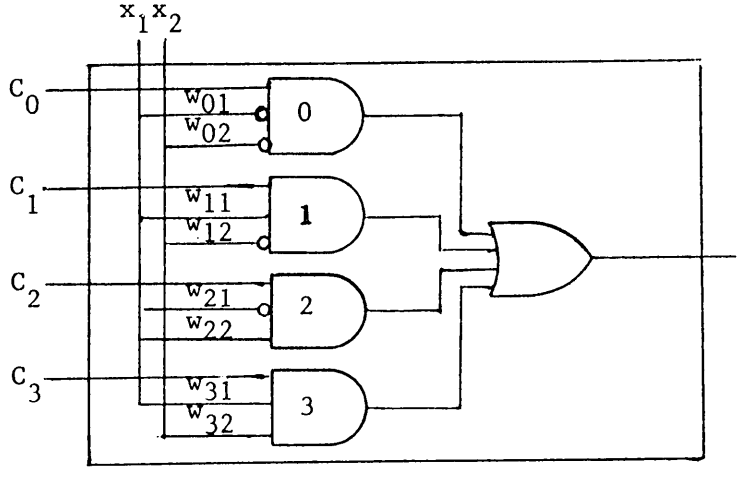

Figure 13: Cell with Two Input Variables x_1 and x_2 for Dislaying Notations C_i, w_{ik} and x_k in detail

Faults	Detection Condition
Group 1: C_i SC C_j	Let $C_i \neq C_j$ and input either $[i]_{10}$ or $[j]_{10}$ depending on whether C_i or C_j is 1
Group 2: C_i BASC C_j	Let $C_i \neq C_j$ and input $[w]_2 = [i]_{10}$.
Group 3: C_i SC x_k	
1. w_{ik} is not inverted to gate i	$C_i=0$, $C_\ell=1$, input $[w]_2 = [i]_{10}$
2. w_{ik} is inverted to gate i	$C_i=1$, input $[w]_2 = [i]_{10}$
Group 4: C_i BASC x_k	
1. w_{ik} is not inverted to gate i	$C_i=0$, input $[w]_2 = [i]_{10}$
2. w_{ik} is inverted to gate i	$C_i=1$, input $[w]_2 = [i]_{10}$
Group 5: w_{ik} BASC C_j	
1. i = j and w_{ik} is not inverted to gate i or if i≠j and w_{ik} is inverted to gate i	$C_i=1$, $C_\ell=0$, $C_j=0$(if i≠j), input $[w]_2 = [\ell]_{10}$
2. i = j and w_{ik} is inverted to gate i	$C_i=1$, input $[w]_2 = [i]_{10}$
3. i ≠ j and w_{ik} is not inverted to gate i	$C_i=1$, $C_j=0$, input $[w]_2 = [i]_{10}$
Group 6: w_{ik} BASC x_h	
1. w_{ik} is inverted but w_{ih} is not inverted to gate i	$C_i=1$, $C_\ell=0$, input $[w]_2 = [i]_{10}$
2. w_{ik} and w_{ih} are both or both not inverted to gate i	$C_i=1$, $C_\ell=0$, input $[w]_2 = [\ell]_{10}$

Group 7: x_k SC x_h	Let C_i=1 where w_{ik} is inverted to gate i but w_{ih} is not and input $[w]_2 = [i]_{10}$.
Group 8: x_k BASC x_h	same as that of group 7
Group 9: x_k BASC C_i	
1. w_{ik} is not inverted to gate i	C_i=0, C_ℓ=1, input $[w]_2 = [i]_{10}$
2. w_{ik} is inverted to gate i	C_i=1, input $[w]_2 = [i]_{10}$

Proof:

Group 1: Since $C_i \neq C_j$ then they will both stay at a low voltage level (logic 0). When input is $[i]_{10}$(or $[j]_{10}$) if C_i (or C_j) is set to logic 1, the error will be detected.

Group 2: Since C_i always follows C_j and the input is $[w]_2=[i]_{10}$, the output will be the logic value of C_j but the original assigned value of C_i is not equal to that of C_j. Error is detected.

Group 3:

1. Since $w_{\ell k}$ is inverted to gate ℓ and C_i is set to 0 and shorted to x_k, then $w_{\ell k}$ will be held low even when x_k is high. That means the ℓth gate will output 1 (so the cell) when the input is $[i]_{10}$. It is an error.
2. Since w_{ik} is inverted to the gate i, the necessary condition to output 1 will be $x_k = 0$. However, C_i is shorted to x_k, so C_i is pulled low when x_k=0 and the gate output will be 0,

(so is the tree output under single fault assumption), when input is $[i]_{10} = [w]_2$. It is an error.

Group 4:

1. Input $[w]_2 = [i]_{10}$, will keep C_i high, so the output of the gate will be 1. It is an error.
2. Input $[w]_2 = [i]_{10}$, will pull C_i low, so the output will be 0. It is an error.

Group 5:

1. If $i = j$, C_i will keep w_{ik} high and the output will output 1, even when $C_\ell = 0$ and input $[w]_2 = [\ell]_{10}$ where x_k is at low voltage. It is an error.

   If $i \neq j$, since $C_j = 0$ will hold w_{ik} low and gate i will output 1 when $C_\ell = 0$, $C_i = 1$ and input is $[w]_2 = [\ell]_{10}$. It is an error.
2. If $i = j$, C_i will hold w_{ik} high and make the output of gate i 0 when $C_i = 1$ and input $[w]_2 = [i]_{10}$. It is an error.
3. If $i \neq j$, $C_j = 0$ will hold w_{ik} low (logic 0), so the output of the gate i will be 0 when $C_i = 1$ and input $[w]_2 = [i]_{10}$. It is an error.

Group 6:

1. It's equivalent to x_{ik} BASC to x_{ih}. If one of them is inverted to gate i and the other is not, any value on x_h (or x_k) will eventually turn off the ith gate by either x_{ih} or x_{ik}. So input $[w]_2 = [i]_{10}$ and set $C_i = 1$ to detect the

error.

2. No matter what x_k is, $w_{ik} = w_{ih}$. So, $C_i = 1$, $C_\ell = 0$, and input $[w]_2 = [\ell]_{10}$ will cause the output of the cell to be 1. It is an error.

Group 7: Either w_{ik} is inverted to gate i and w_{ih} is not or vice versa. Input $[w]_i = [i]_{10}$ will always turn off gate i even when $C_i = 1$. It is erroneous.

Group 8: as the proof of Group 7.

Group 9:

1. $C_i = 0$ will set $w_{\ell k} = 0$, that is, an 1 input to gate ℓ all the time. Therefore $C_\ell = 1$ even with input $[w]_2 = [i]_{10}$ will cause the cell to output 1. This is an error.
2. $C_i = 1$ will always turn off gate i, even when the input is $[w]_2 = [i]_{10}$ and $C_i = 1$. This is an error.

For a single cell, a detection set which is able to detect any R fault have to provide all the above detection conditions to the cell. The multiple s-a-f's detection test is not adequate for this job. For instance, in Fig. 13, C_0 short to C_3 will not be detected by that test. By using both multiple s-a-f's detection test and s-a-f location test while inputing all possible input combinations to the cell will do the job. But, for detecting an R fault in a modular tree, the work of propagating those (i,k)-patterns to each cell becomes too tedious and hopefully can be avoided.

However, the test, designed by Cioffi and Fiorillo [11], for detecting unrestricted faults in a tree shown in Fig. 1a, can detect

any single R fault in that tree also. The testing procedure is reviewed briefly as follows:

1. Fixed 0 Test: Set all the control leads to 0. Input all the 2^n binary n-tuples at the primary input wires and observe the outputs.
2. Fixed 1 Test: Set all the control leads to 1. Input all the 2^n binary n-tuples at the primary input wires and observe the outputs.
3. Traveling 0 Test: Set all the control leads except the ith control lead to 1 and the ith control lead to 0. Input a binary n-tuple corresponding to the binary equivalent of i at the primary input wires and observe the output. Do this step for every i, $0 \leq i \leq 2^n-1$.
4. Traveling 1 Test: Set all the control leads except the ith control lead to 0 and the ith control lead to 1. Input a binary n-tuple corresponding to the binary equivalent of i at the primary input wires and observe the output. Do this step for every i, $0 \leq i \leq 2^n-1$.

In the above procedure, it takes 2^{n+1} tests to detect faults in a combinational tree but note that, this procedure involves $2\cdot(2^n+1)$ patterns, two of them for the first two tests and the rest for the last two tests. The new test designed in this chapter may take more tests for detecting R faults in a tree with n input variables, but, the number of patterns needed is much smaller than $2\cdot(2^n+1)$. Also, the test conditions can be easily derived as described in the next section.

4.2 $N(m_M)$-pattern and Fault Detection Algorithm A and B

Before actually constructing the test set, some special control patterns are introduced below.

Definition:

An m_M-pattern (where $1 \leq M \leq 2^m$, $m \in N$) has length 2^m; its Mth bit is one and all other bits are zero's.

Definition:

An $N(m_M)$-pattern (where $2 \leq N$, $1 \leq M \leq 2^m$, $m \in N$) has length $2^{N \cdot m}$ and is formed by cancatenating $(N-1)(m_M)$, $(N-1)(m_{M+1})$,.....$(N-1)(m_{2m})$, $(N-1)(m_1)$,.....$(N-1)(m_{M-1})$ together.

Example:

2_1-pattern: 1000 2_2-pattern: 0100 2_3-pattern: 0010

2_4-pattern: 0001

$2(2_1)$-pattern: $2(2_1) = (2_1)(2_2)(2_3)(2_4)$

$= 1000010000100001$

$2(2_3) = (2_3)(2_4)(2_1)(2_2)$

$= 0010000110000100$

Note that, when m=1, the only two $N(1_M)$-patterns, $N(1_1)$- and $N(1_2)$- patterns, are exactly the two (i,1)'-patterns where N=i+1.

Fault Detection Algorithm A (For detecting any single R fault in a cell with m input variables)

Let M = 1

I. Set m_M-pattern to the cell and input $[w]_2 = [(M-1)]_{10}$ and $[(M-1)^*_k]_{10}$ for each k where $1 < k < m$. (The cell should output 0's

except when the input is $[(M-1)]_{10}$.)

II. Let M = M+1; if M $\leq$ m go to Step I, otherwise terminate the procedure.

It is easy to see that this test will detect any single R fault in a cell. Since if we input $[w]_2 = [(M-1)^*_k]_{10}$ for $1 \leq k \leq m$ under each m_M-pattern we satisfy detection condition 3(1), 4(1), 5(2), 6(2) and 9(1). All the rest will be taken care of by inputting $[w]_2 = [(M-1)]_{10}$.

Note that the above test can also detect any single s-a-f in a cell, because it obviously provides all detection conditions for those s-a-f's mentioned in Lemma 4 and Theorem 1.

Since we can detect an R fault in a cell, the only problem left for designing a detection test for a whole modular tree is how to propagate m_M-patterns efficiently to those cells inside the tree. In other words, repeating algorithm A for each cell in the tree should be avoided. Patterns which will propagate m_M-patterns to every cell inside the tree by inputting proper input combinations need to be found. Fortunately, the $N(m_M)$-patterns are just adequate for that job. And, if $N(m_M)$-patterns are used, the number of patterns needed for detecting an R fault in a tree only depends on m, where m is the maximum m_h. More clearly speaking, if the biggest cell in a tree has m input variables, 2^m is the number of patterns needed for detecting an R fault.

The algorithm for detecting an R fault in a modular tree with n input variables is given below.

Fault Detection Algorithm B

(m: the maximum m_h, refer to Fig. 1a)

Let M = 1

I. Input all possible input combinations to the tree under the $N(m_M)$-pattern. (N is chosen such that $(N-1)\cdot m \leq n \leq N\cdot m$.)
(If $2^{N\cdot m} > 2^n$, use the first 2^n bits of the $N(m_M)$-pattern.)

II. Let $M = M+1$. If $M \leq m$ go to Step I, else terminate the procedure.

As mentioned previously, the two $N(1_M)$-patterns are exactly the two $(i,1)'$-patterns where $N = i+1$. Therefore, for a binary tree ($m_h = 1$ for all h as shown in Fig. 1a) the multiple s-a-f's test procedure can detect any single R fault at the same time. Nevertheless, the capability of the above test algorithm to detect single s-a-f in a combinational tree shown in Fig. 1a should not be ignored.

The number of tests needed to detect an R fault in both combinational and definite trees is formulated as follows:

1. For a combinational tree shown in Fig. 1a, the number of tests for detecting an R fault is $\underline{2^m}\cdot 2^n$
   where m is the maximum number among m_h
   n is equal to $\sum_{h=1}^{k} m_h$.
2. For a definite tree shown in Fig. 11a, by the same argument given in Section 3.2, instead of input combinations to a combinational tree, a time sequence of inputs has to be applied to the definite tree. For a tree shown in Fig. 11a, a sequence which contains as subsequence all k_d-ary sequences of length d has length $2^{k_d\cdot d}+d-1$. However, the number of patterns needed in this case is dependent only on k_d and is 2^{k_d}.

Therefore, the number of tests for detecting an R fault in a

definite tree shown in Fig. 11a is $\underline{2^{k_d}} \cdot (2^{k_d \cdot d} + d - 1)$.

The underlined part of both formulae indicates the number of patterns used during the detection test.

5. CONCLUSIONS

5.1 Summary

In this thesis, several algorithms for either fault location or fault detection are presented. Three fault models, stuck-at faults, short circuit faults and broken and short circuit faults are involved in the discussion. But, only for the first model, s-a-f's, is the location problem studied. Algorithms for locating single (real or pseudo) s-a-f's in either a faulty cell or a faulty tree are presented. For the other two models, Detection Algorithms for faults in a restricted sense are designed. Because of the special structure of a modular tree, appropriate selection of control patterns becomes the main work of designing a test for either fault detection or fault location in the tree.

In Chapter 2, special detection conditions for single s-a-f's in a cell are introduced in Lemma 4 and Theorem 1 and special control patterns, (i,k)- and (i,k)'- patterns, are defined. An algorithm, Location Algorithm I, for locating single s-a-f's in a cell is designed by using (i,k)-patterns.

Location Algorithms II and III for fault location in a combinational tree under either single or multiple s-a-f's assumption are given in Chapter 3. Awβ form, the most inportant tool for fault location in a tree, is defined there also. The results of Lemma 5 and Theorem 2 prove the locatability of s-a-f's which simulate a single s-a-f under the multiple s-a-f's detection test. So, in fact, Fault Location

Algorithm III is capable of locating multiple s-a-f's to some extent. All the results obtained from combinational trees can be applied to definite tree directly. Even though, the detailed algorithm is omitted, upper bounds of the test length for definite trees with depth d under single or multiple s-a-f's assumption are developed in table form.

In Chapter 4, faults which can be detected by Detection Algorithm A and B are restricted to the R faults. Special detection conditions for R faults are studied first, then N(m)-patterns are designed for detecting single R fault in a cell and a combinational tree respectively. Also, those results can be applied directly to the defenite tree.

5.2 Suggestions for Future Study

1. Extension of the Fault Location Algorithm III to locate multiple s-a-f's not restricted to the pseudo-single s-a-f in Modular Trees.

2. To find the whole fault set which can be detected by the Fault Detection Algorithm B.

REFERENCES

1. B. A. Prasad and F. G. Gray, "Fault Diagnosis in Uniform Modular Realizations of Sequential Machines," Digest of Papers 1973 International Symposium on Fault Tolerant Computing, Palo Alto, California, June, 1973.

2. B. A. Prasad and F. G. Gray, Research Initiation: A Theoretical Investigation of Diagnosable Logic Systems, Final Report, National Science Foundation, Washington, D. C., October, 1973 (Grant 6J-32718)

3. B. A. Prasad, Multiple Fault Detection in Interactive Logic Structures, Virginia Polytechnic Institute and State University, Doctoral Dissertation, April 1974.

4. S. W. Golomb, Shift-Register Sequences, Holden-Day, San Francisco, 1967.

5. S. S. Yau & C. K. Tang, "Universal Logic Modules and Their Applications," IEEE Transactions on Computers, Vol. C-19, pp. 141-149, February 1970.

6. T. F. Arnold, Universal Structures with Uniform Interconnection Patterns for Synchronous Sequential Circuits, Columbia University, Doctoral Dissertation, 1969.

7. T. F. Arnold, C. J. Tan, and M. M. Newborn, "Iteratively Realized Sequential Circuits," IEEE Transactions on Computers, Vol. C-19, pp. 54-66, January 1970.

8. A. D. Freidman, "Feedback on Synchronous Sequential Switching Circuits," IEEE Transactions on Electronic Computers, Vol. EC-15, pp. 354-367, June 1966.

9. F. G. Gray and R. A. Thompson, "Reconfiguration for Repair in a Class of Universal Logic Modules," IEEE Transactions on Computers, Vol. C-23, pp. 1185-1194, November 1974.

10. W. W. Peterson and E. J. Weldon, Jr., Error-Correcting Codes, MIT Press, Cambridge, Mass., 1972.

11. G. Cioffi and E. Fiorillo, "Diagnosis and Utilization of Faulty Universal Tree Circuits," AFIPS Conference Proceedings 1969 Spring Joint Computer Conference, pp. 139-147.

12. S. Ramo & A. R. Smith III, "Fault Detection in Uniform Modular Realizations of Sequential Machines," Digest 1972

International Symposium on Fault Tolerant Computing, Newton, Massachusells, June 1972, pp. 114-119

13. S. Ramo, Technical Report: Fault Tolerant Uniform Modular Decompositions of Sequential Machines, National Science Foundation, May 1972, (Grant GK-5406).

14. K. C. Y. Mei, "Bridging and Stuck-at Faults," Digest 1973 International Symposium on Fault Tolerant Computing, Palo Alto, California, June 1972, pp. 91-94.

15. L. C. Shih, F. G. Gray and R. A. Thompson, "Fault Location in Modular Trees," Proceedings of the Annual Southeastern Symposium on System Theory, April, 1976, pp. 238-243.

The vita has been removed from the scanned document

FAULT DETECTION AND LOCATION IN MODULAR TREEW

by

Lionel C. C. Shih

(ABSTRACT)

The problem of fault detection and location in modular tree structures are considered in this thesis. The fault set is restricted to stuck-at faults in the discussion of the fault location problem. Short circuit faults and broken and short circuit faults are considered in the discussion of the fault detection problem.

In either the fault detection or location case, detection conditions for each fault in a cell are derived. Tests for locating or detecting fault are designed based on providing those detection conditions to each cell in a tree. For fault detection, the more detection conditions provided by a test, the better the test is. The detection conditions provided by the detection test must be partitioned into single element blocks to locate faults. In this thesis, two algorithms for fault detection and three for fault location in a combinational modular tree with n input variables are presented in detail. All the above results are directly modified for fault detection and location in modular trees which realize arbitrary definite machines. Since a pair of these tree structures can be connected to realize arbitrary sequential machines, the results derived here are useful in diagnosing sequential machines.