AN ABSTRACT OF THE THESIS OF

Chandramouli Ramamurti for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on May 5, 1978

Title: MULTIPLE-FAULT DETECTION IN ITERATIVE ARRAYS

Redacted for Privacy

Abstract approved:

With the advent of LSI, iterative forms of realization of digital systems are becoming increasingly popular with system designers. Many problems occurring in digital computer design render themselves suitable for iterative realization. These include adders, arithmetic logic units, coding and decoding circuits and so on. Fault-free functioning of such systems is very important, and this dissertation develops multiple-fault detection tests for the above class of arrays: namely, one and two dimensional iterative arrays. The difference from previous work is that each cell is modeled in terms of its state graph behavior as opposed to more conventional techniques, which emphasize the sensitization of paths representing the electronics of the realization.

Using the transition matrix representation of the flow table of an arbitrary cell in the array, the fault detection test is generated by an algorithm that involves the compar-

ison of the rows in the matrix corresponding to the normal and faulty state-behavior of the cell. The tests generated are such that they test simultaneously more than one cell in the array for the given fault. A graph-theoretic condition is imposed on the state graph of the cell, for the existence of such tests. The case of two-dimensional array is viewed as equivalent to a one-dimensional array by compression, either horizontally or vertically. This enables the extension of the fault-detection test algorithm for the one-dimensional array to the two-dimensional array without further modification.

Multiple-Fault Detection in Iterative Arrays

by

Chandramouli Ramamurti

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed May 5, 1978

Commencement June 1979

APPROVED:

Redacted for Privacy

Professor of Electrical and Computer Engineering
in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented May 5, 1978

Typed by Lora Wixom for Chandramouli Ramamurti

ACKNOWLEDGMENT

I would like to express my sincere appreciation to Dr. R. A. Short, a dedicated educator, for his patience, guidance and suggestions throughout my graduate career, and above all, for having inculcated in me the tenacity to persevere. Further gratitude is expressed to all the members in my doctoral committee, and special appreciation goes to Dr. W. R. Adrion who helped me with various, valuable suggestions for my thesis. Sincere thanks are due to Dr. R. R. Mohler for having provided me with financial assistance throughout my stay at O.S.U.

Furthermore, I offer my sincere gratitude to Kalpa and Sri who were a constant source of moral support during my stay in Oregon.

Last, but not least, I owe everything to my father, mother, brothers and sister, without whose constant support I could not have achieved these goals.

TABLE OF CONTENTS

LIST OF FIGURES

MULTIPLE-FAULT DETECTION IN ITERATIVE ARRAYS

CHAPTER I

INTRODUCTION

Section 1.1

The current upsurge in the use of computing machinery in all walks of life has made more crucially important the error free operation of such machines. Von Neumann makes an analogy to nature wherein he says, "...the basic principle of dealing with malfunction in nature is to make the effect as unimportant as possible and to apply corrections if they are necessary at all, at leisure,... ." In the case of computers one cannot usually afford leisure, but diagnosis should be immediate in the event of malfunction, as otherwise it would lead to chaos in extreme environments such as on-board computers in space vehicles.

As a result, considerable research has been directed towards the area of fault-tolerant computing over the past decade. The work can be broadly divided into two sub-areas; namely, fault diagnosis and fault-tolerant redundancy techniques. The former is most important in the day-to-day maintenance of computing equipment while the latter is of utmost importance in places where human interaction with the machine is not possible, such as space vehicles.

Of course the most important factor in the design of

reliable aerospace computers is to manufacture each component, each part and each subassembly, so that they are highly reliable. Because of increased performance requirements and longer mission duration, it is still necessary, however, to devise systems that tolerate failures. For example, the Saturn-V booster of the Apollo moon mission used a computer with two-out-of-three voting redundancy, implemented at the logic gate level, throughout the machine. This approach results in a three-fold increase in the component count, but significantly enhances the reliability. Similarly the lunar excursion module (LEM) was equipped with a backup guidance system, which provided for LEM vehicle control in the event of failure of the primary system.

However, it is evident that one need not usually resort to such costly techniques in the case of systems where direct human intervention is possible at all times. It is in such systems that fault diagnostic techniques play a key role in the day-to-day running as well as in the manufacturing of these systems. After detecting the presence of a fault in a circuit, location of that fault up to a cell or component within the circuit involves further testing. This results in increased testing time and cost. Due to mass production at the chip level, a mere detection of the presence of fault will be sufficient and the faulty chip can be replaced with a good chip. We

thus avoid further testing of the faulty chip in order to locate the fault within that chip. Hence we see that fault detection plays a key role in most diagnostic procedures.

The literature abounds with various fault-diagnostic techniques for both combinational and sequential circuits. However such techniques are not efficient when applied to iterative circuits, which are highly structured circuits, frequently used by engineers in the design of digital systems. This is due to the fact that iterative circuits have a regular geometry as compared to general combinational and sequential circuits which are not regular in geometry.

In this dissertation we shall address ourselves to the problem of fault detection in iterative combinational circuits. Since iterative circuits can be packaged within a single chip, the fault diagnostic procedure can be limited to just fault detection. The following section will consider the nature of this problem. Then we shall review briefly the salient contributions that have been made in the field of fault tolerant computing. We shall stress the relevance to the complex digital systems of today and tomorrow and also attempt to put our particular problem into perspective.

Section 1.2

Rapid progress in LSI electronics suggests that circuits possessing a high degree of regularity will become

increasingly attractive because of their relative ease of fabrication. This has motivated considerable research into the design of regular arrays of logical devices to perform such functions as coding and decoding (Levitt and Kautz [22]), data switching (Kautz [19]), and more general logic operations (Kautz [18], Minnick [30], [31]). A large number of duplicate integrated circuits can be fabricated on single silicon wafer using the LSI technology.

There are two ways one could obtain the desired functions with such wafers. In one way, the duplicate circuits can be separated, packaged in separate chips and interconnected to form the desired functional circuit. The interconnections need not necessarily be uniform. On the other hand, each of these duplicate circuits can be uniformly interconnected with its neighboring cells on the same wafer according to some regular pattern. The duplicate circuits are called cells, and the entire set of uniformly-connected, duplicate circuits are called a cellular or iterative array. From the latter type of interconnection, considerable reductions can be made in circuit wiring and the number of package leads, thereby leading to higher reliability and lower cost.

However the techniques for fault diagnosis differ considerably from those for non-iterative forms of circuits. This is due to the fact that the effect of a fault in an arbitrary cell within the array is visible only at the

boundaries of the array, and testing of the array for this fault involves exercising the corresponding cell deep inside the array - possibly through all possible input combinations.

Kautz [17] was the earliest to investigate this problem under a generalized fault assumption - the faults due to malfunction of components within the cell may cause an arbitrary change in one or more outputs of the cell (external outputs or outputs to adjacent cells - although he assumed the occurrence of single faults only).

Kautz's work was followed by Friedman [12] who analyzed the iterative array - both one and two dimensional - under a restricted fault assumption, i.e; that only one output variable can change at a time, and only a single faults occur. In either of these approaches the notion has been to classify special types of cells; e.g., cells with flow tables that have permutation columns (each state appears only once in a column in the flow table and all the states appear in a column) for which tests can be generated easily.

Prasad [35], too, has proposed similar ideas in his paper, and he proves that the existence of the permutation column in the flow table leads to an easier testing process; in fact, methods of deriving test sets for such flow tables require trivial computations. Recently, Dias [9], using the concept of a distinguishing sequence (Hennie [15]),

has described a test generation procedure, wherein each test is capable of detecting faults simultaneously in more than one cell of the one dimensional array. The tests, called loop tests, can be used only with one-dimensional iterative arrays with each cell having external outputs. The author has shown that the problem of testing a one-dimensional array for multiple faults is equivalent to verifying the correctness of the truth table of the entire array. It is further proved that the test set generated is sufficient to verify the correctness of the truth table of the array.

Turning to fault detection procedures for two-dimensional iterative arrays, work done so far (Kautz [17], Friedman [12], Landgraff [21], Chia [7], Prasad [35], Dias [9]) has concentrated on determining a set of tessellations (a set of inputs that can be applied to a cell in the two-dimensional array) such that the tessellation set contains all possible inputs that can be applied to a cell input. It has been found that such a set doesn't exist for all arrays. All have assumed single faults in their analysis, except for Prasad and Dias.

Thus, multiple-fault detection in iterative arrays has not been given much attention except by Dias [9] and Prasad [35]. Test sequences that are cyclic in nature; i.e., that test more than one cell at a time in the given array, can be exploited in order to reduce the number of

tests. There hasn't been any work in the generation of such cyclic tests except for Dias, whose arrays need external outputs from each cell in order to achieve such tests.

It is the intent of this author to delve further into multiple fault analysis of both one and two-dimensional arrays and to investigate the nature of cyclic test sequences for arrays without external outputs from each cell. We shall examine the existence of such tests using a graph theoretic condition. The nature of faults assumed in our case will be similar to that of Kautz, in that the cell output will be allowed to change into any of the (n-1) faulty states (for an n state flow table which describes the cell), an assumption that potentially covers a much wider class of faults than usually assumed.

In the case of a two-dimensional array, a completely new approach has been taken: the two-dimensional array is considered as a one-dimensional array where each cell (modified) performs the function of all cells in the corresponding row (or column) in the two dimensional array. This enables the extension of the one-dimensional-array test algorithm to the two-dimensional array, with only slight modifications.

Section 1.3

Fault diagnosis emerged as an important problem in the 1950's when computers first appeared as fully developed digital systems. Many of these systems relied on a combination of skilled maintenance and special hardware to detect and locate faults. But as the complexity of the logic circuitry increased, this approach tended to be impractical because of its reliance on the skills of technicians, and different approaches were necessary. An early contributor was Eldred [10], one of the pioneers who came up with a new idea of testing; namely, that of testing the machine hardware rather than its functions. This was the first time that anyone came up with a set of tests for testing a combinational circuit. In his paper he discussed methods to find input sequences to AND-OR circuits which allow the detection of faults by observation of its outputs to specific test input sequences. His work formed the basis for the path sensitization methods to be proposed later by Armstrong [3] and Roth [36].

In order to proceed further it will be helpful if certain terms are defined. Fault Detection is the process of applying a set of tests to a circuit and deducing the presence or absence of faults in the circuit by observing the output of the system. Fault Location is a refinement which enables the localization of the fault to a module or

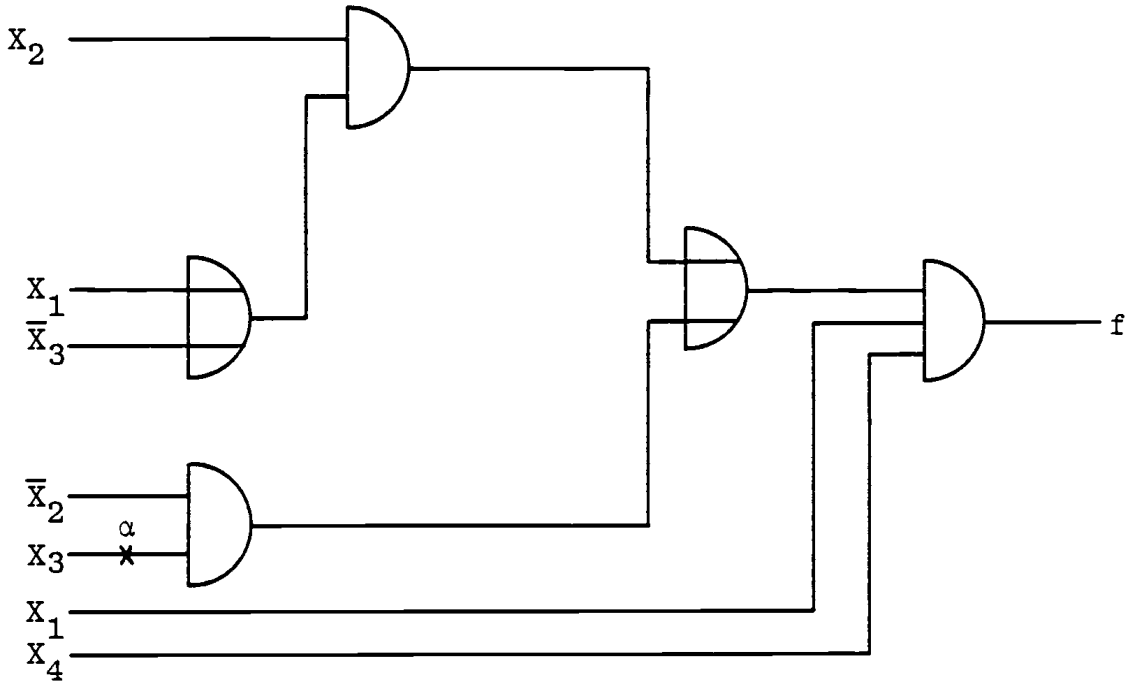
component. Finally, Fault Diagnosis includes either or both of these processes. In general we are interested only in faults that alter the logic function of a digital circuit. These may be caused by such as an open circuit on the network leads, a shorted diode or perhaps a short circuit of the network leads. (Under such conditions the logical value on those faulty leads are said to be equivalent to either a logical '1' or logical '0'. In the former the lead is said to be stuck-at-1 written as 's-a-1', and in the latter stuck-at-o written as 's-a-0'.

However with the increasing complexities of the present IC chips, the identification of "stuck-at" faults for a particular component or connection seems irrelevant. This is because one is concerned only with whether the entire chip is faulty or not, rather than with the particular nature of the fault inside the chip. Due to mass production, it is less costly to replace a faulty chip than to identify, locate and rectify the particular fault inside the chip.

Faults may be either permanent or intermittent. Once having occurred, the permanent fault will remain until the faulty component is repaired or replaced. The intermittent fault is of a transient nature, causing only momentary changes in the circuit outputs. Recent studies (Mei [27]) have come up with a new type of fault known as a 'bridging fault' (permanent), which is a fault caused by unwanted

conducting paths, such as a spurious path between two leads in a finished IC chip. Such faults are said to cover a wider area of malfunction especially in the IC chips of present day.

Let us illustrate how faults can affect a given logic circuit. Consider the Circuit in Figure 1. Let us assume the wire 'a' be s-a-1; i.e., the variable X_3 is s-a-1. Due to this fault the output function f will not be the function acutally realized. The corresponding AND gate (whose inputs are $\bar{X}_2$ and X_3) will have an output $\bar{X}_2$ as long as X_3 is s-a-1 resulting in a faulty output function $f = (X_2 (X_1 + \bar{X}_3) + \bar{X}_2) X_1 X_4$. In order to detect the presence of this fault we have to determine a suitbale input combination, called a test sequence, which, when applied to the circuit, will produce an output different from the output of the circuit under fault-free conditions. However this is not possible for all circuits. In redundant circuits the presence of certain faults will not affect the normal functioning of the circuit (Friedman [11]). As an example, consider the redundant circuit in Figure 2. Irrespective of the nature of fault on the lead α, corresponding to the variable Z, the circuit will generate a fault free function. Moreover a fault on α(S-a-o or S-a-1) is not detectable.



$$f = (X_2(X_1+\bar{X}_3)+X_3\bar{X}_2)X_1X_4$$

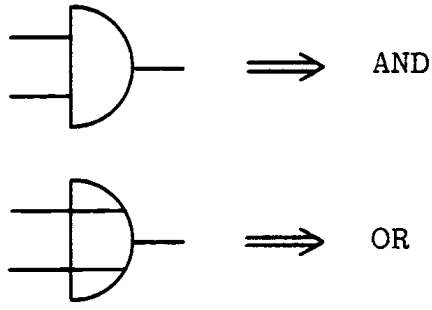


Figure 1. An example of a combinational circuit with a single fault.

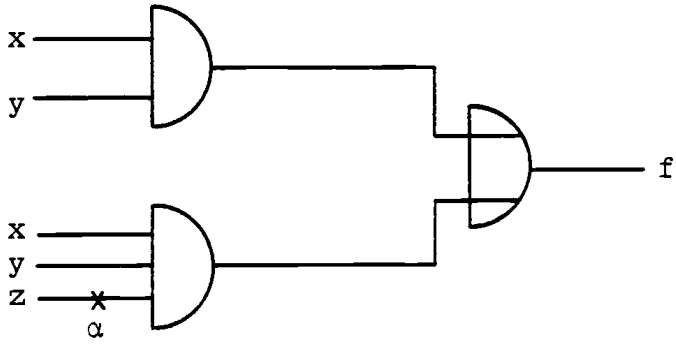


Figure 2. Redundant Circuit

As mentioned earlier, the path sensitization method was one of the earliest concepts in tackling the fault detection problem. The basic idea is to choose a path from the site of failure on to a circuit output and to "sensitize" this path. Sensitizing a path means assigning logical values to gates on this path such that the output of the circuit depends solely on the fault that has occurred. Sensitizing each path from failure site to output becomes a cumbersome procedure for larger circuits, and this led Roth [36] to generalize the method in such a way that all possible paths from the site of failure to the circuit output are sensitized simultaneously, and concurrently a test derived to detect that fault. Roth called this process the D-Algorithm, which happens to be the first practical combinational circuit testing procedure. It is also one of the widely used because of its versatility.

While Roth's contribution tended to be more practical and widely used, there were also a host of other investigations in this area. Most of these are of pedagogic interest only, although they did contribute to a better understanding of the basic concepts in fault analysis. Poage [34] was one of the first to derive tests using the functional description of the circuit. The functional description represents not only the relationship of the output variable to the signals on the input and internal

lines, but also the effect of fault on the output. This is in contrast to the earlier methods where tests were generated by assuming faults in each component within the circuit and then propagating them to an external output by using the path sensitizing concept.

Armstrong [3] followed Poage's concept, but altered his method so that the functional descriptions involved the identity of individual gates in the circuit. Yet another approach was due to Sellers and Hsiao [37] who used the Boolean difference of the circuit in order to analyze the effects of a fault on the network output. (The Boolean difference is the exclusive - OR sum of the faulty and normal output of the circuit, hence its value shows whether the circuit is fault-free or not).

The engineering cost question arises in these problems, and in this context cost reflects the number of tests; i.e., a minimal test set has least cost. Using mapping techniques and a tabular representation - akin to the prime implicant table (McCluskey [26], Kohavi [20]) derived minimal test sets for combinational logic circuits. Schertz and Metze [28] introduced the technique of "fault collapsing" to reduce the number of faults to be considered, thereby reducing the number of tests. (Faults which have identical test sets form an equivalence class, and it is sufficient to derive tests for just one member of that class. For example given an AND gate a test to detect

S-a-0 on the input leads will be the same as the one for S-a-o on the output lead. These S-a-o faults on the input and output leads are thus indistinguishable and each fault need not be tested individually. Such a technique is known ad fault collapsing.)

Most of the techniques discussed above have been concerned with the diagnosis of single faults. However such an assumption may be quite improper for current LSI technologies where faults are very much dependent upon one another. In such cases a multiple-fault assumption (more than one fault occurring at the same time) is probably necessary to produce reasonable detection results. Designing multiple-fault detection tests for a combinational circuit is computationally complex since the number of multiple faults is larger than single faults even for circuits with a small number of inputs. We have (3^p-1) multiple faults in a 'p' line network. Hence efforts have been directed towards partitioning multiple fault into equivalence classes which result in lesser number of tests to be derived for fault detection purposes.

Several approaches to this multiple fault problem have been proposed in the literature and we will enumerate only the salient features of these studies. Test sets derived under the single-fault assumption have been proven to be capable of detecting multiple faults too (Metze and Cha [28]). Many more authors have looked into the problem,

notable among them being Yau and Tang [46], Bossen and Hong [5], Schertz and Metze [29]. The methods suggested by these authors involve a generalization of the fault collapsing technique; i.e., the utilization of equivalence classes of faults, as mentioned earlier.

We have thus far discussed the usual approach of detection involving constant human interaction for the fault-free operation of the system. However this has proved to be inefficient in many cases, because of unacceptable delays involved in real time programs, due to manual repair action, inaccessability of some systems for repair, and the high cost of maintenance. An alternate approach which alleviates much of these shortcomings is provided by fault tolerant redundancy techniques. These use the design of the network itself to act as self-checking or self-correcting, thereby avoiding human intervention. A system is fault tolerant if it functions properly (or executes its program correctly) despite the occurrence of logic faults. Fault tolerance is introduced into a system so that the reliability and availability of the system is increased.

Redundancy techniques have found extensive application in spaceborne computers and in intensive health care instrumentation. An extensive and exhaustive summary on research in this area can be found in Short [40] and Avizienis [4].

A more detailed review of pertinent work done in the

area of fault diagnosis of iterative combinational array appears in Chapter II, followed by a brief discussion of the essential goals of this dissertation. Chapter III deals with fault detection in the one-dimensional array (combinational), while Chapter IV deals with two-dimensional iterative (combinational) arrays. A fault model has been developed in Chapter III, followed by an algorithm for test generation, and finally some examples are given to illustrate the techniques. The same algorithm is extended to the two-dimensional iterative array in Chapter IV, and an example is presented to illustrate the algorithm. Finally, Chapter V brings out the main conclusions and consequences of the dissertation in the form of a summary, with suggestions for future work.

CHAPTER II

REVIEW OF PREVIOUS ITERATIVE ARRAY RESULTS

Designing switching functions with iterative cells offers considerable flexibility and practical physical layout advantages compared with general combinational designs that realize the same switching function (McCluskey [25], Hennie [15]). Iterative arrays were considered formally by Hennie [15] in 1961 and have since attracted many researchers into the area of iterative array design.

Many problems occurring in digital system design render themselves suitable for iterative realization such as adders, arithmetic logic circuits and so on. Minnick [30], Maitra [23], Mukopadhyay [32] have considered the realization of arbitrary switching functions in iterative form, while Nicoud [33] and Cappa [6] have used iterative realizations for arithmetic manipulations such as radix conversion and binary division. Such regular arrays provide the advantage of large gate to-pin ratio and of ease of fabrication, (due to the advent of LSI technology) as has been said earlier. In view of this design concept, error-free maintenance of such circuits requires a careful study of its fault diagnostic aspects.

One of the earliest to examine fault-diagnostic techniques for iterative arrays was Kautz [17]. He analyzed

both one and two dimensional iterative arrays under a single cell fault assumption. However he assumed the cell output to change arbitrarily in the event of a fault in that cell. By this he meant that faults in the cell output could be due to either multiple or single faults within the cell. The signals in the array were assumed to flow from left to right in case of a one dimensional array and from top to bottom, as well, in the case of a two-dimensional array. Such signal flow is defined as a unilateral flow of signals. Kautz's method of analysis is based on two conditions; namely, (1) it should be possible to apply all possible inputs to any cell in the array in order to test the array; (2) for each fault in a cell there should be a sensitized path which transmits the effect of a fault to an observable output. He proves that a one-dimensional array can be tested with a minimal number of tests (mn) (m is the number of columns and n the number of rows in the flow table) if each state appears an equal number of times in the table and no column has two like entries; i.e., if all columns are permutation columns. As an example, a parity checker represented by the following flow table can be tested with m x n = 2 x 2 = 4 tests irrespective of the number of cells in the arrays. Fault location is possible only if each cell has an external output. In order to locate the fault within a pair of

x \ y	0	1
A	A	B
B	B	B

Flow table for parity checker.

adjacent cells it is necessary that no two rows be alike in the external output entries in the flow table.

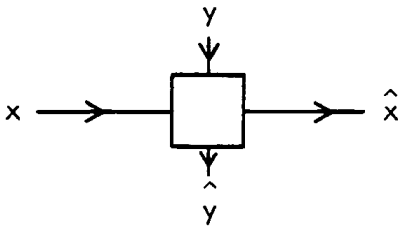
In case of two-dimensional arrays, Kautz restates his earlier conditions to suit the structure of the two-dimensional array. The existence of a set of tessellations covering all possible input combinations is a necessary condition for the array to be testable. (A tessellation of the array is defined as a complete set of cell input states which are configured in the same rectangular pattern of the array and are compatible with one another along the cell interface. By compatible we mean that, each cell output state is the same as the input state to its right and each cell vertical output is the same as the vertical input to the cell below.) Figure 3 will help us understand the concept of tessellation.

The set of states corresponding to the loop ((2, 1), (4, 2)) can be applied to all cells in a two-dimensional array; i.e., the set of inputs can be tessellated in both dimensions (horizontal, and vertical) in a two-dimensional array such that there won't be any incompatibility at cell

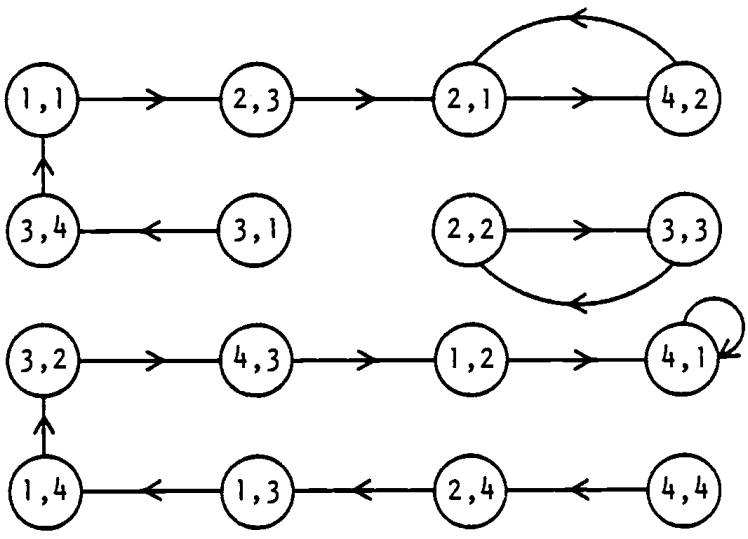
x \ y	1	2	3	4
1	(2,3)	(4,1)	(1,4)	(3,2)
2	(4,2)	(3,3)	(2,1)	(1,3)
3	(3,4)	(4,3)	(2,2)	(1,1)
4	(4,1)	(2,1)	(1,2)	(2,4)

(a) Flow Table

Note: The entries in the flow table are of the form $(\hat{x},\hat{y})$ where $\hat{x}$ is the horizontal output and y is the vertical output of the iterative array cell. (x,y) are the horizontal and vertical input to the cell.



(b) Basic Cell



(c) Successor Graph

Figure 3. Derivation of tessellation set.

interfaces as Figure 4 shows. Hence a tessallation input can be also defined as a set of input states corresponding to a loop in the successor graph of the flow table because of its capability of being tessellated in a two-dimensional array. If the input states corresponding to such loops in a successor graph cover all possible (cell) input combinations, the input states in each of these loops can be applied to all cells in a two-dimensional array. The fault in a given cell will propagate to an external output either in the horizontal direction, or the vertical direction, or both, if no two rows or two columns of the flow table are alike.

The tessellation problem is the same as the domino problem, where a domino is a square tile with different colors on each of its four edges.* Tammaru [42] has extended Wang's undecidability condition to the tessallation problem in two dimensional iterative arrays and in essence states that existence of a set of tessallations covering all the input combinations cannot be proved a priori; i.e., prior to the test generation procedure.

---

*The four edges correspond to the two inputs and two outputs of a typical cell. Each such color pattern on a domino is a domino type. Wang [45] has proved that there doesn't exist a general algorithm which will decide whether an infinite plane can be covered with a finite set of domino types such that any two adjoining edges have the same color.

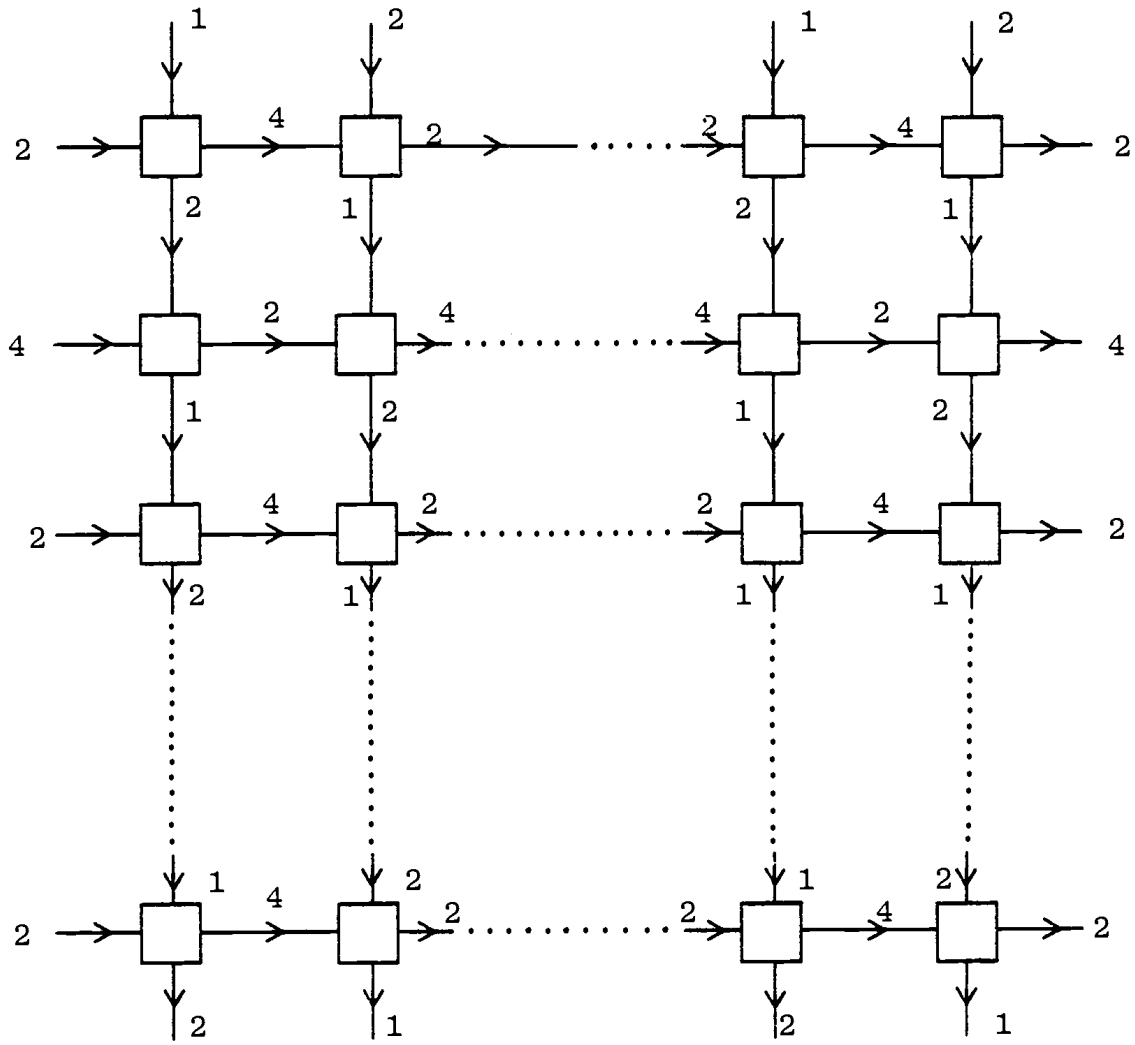


Figure 4. Application of the input combination (2,1),(4,2) to a two-dimensional array.

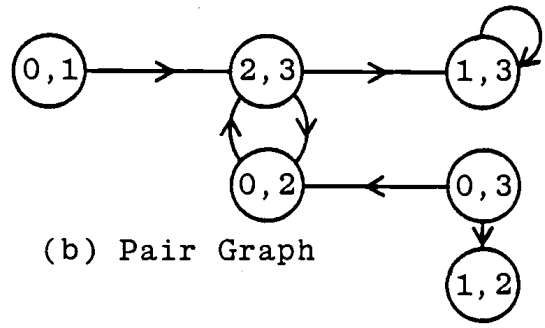
In contrast to Kautz, Friedman and Menon [12] made a restricted fault assumption; i.e., only one horizontal or vertical output lead can be faulty at a time. Friedman proposes that specific set of inputs less than the set of all possible inputs will be sufficient to test the iterative array because of this restricted fault assumption. Using the principles of ultimate distinguishability, necessary and sufficient conditions have been derived for the testing of the one-dimensional array. Two states 's_i,' and 's_j' are said to be ultimately distinguishable if there exists an external input combination to the one dimensional iterative array such that the observable output is different when s_i and s_j are applied to the first cell of the array. In order that there be a test for every fault, the normal and faulty outputs or output states (of the faulty cell) corresponding to that test should be ultimately distinguishable.

States that are ultimately distinguishable can be obtained from a pair graph. A pair graph is one where each node is a pair of input states $(s_i, s_j; i \neq j)$ and contains $\binom{n}{2}$ nodes corresponding to the total number of state pairs in an 'n' state flow table. A directed edge exists between two nodes in the pair graph if there is a transition between the two according to the flow table. The nodes in a closed loop or nodes in a path leading to

a closed loop represent a pair of ultimately distinguishable states.

x \ y	0	1
0	0	2
1	0	3
2	0	3
3	2	1

(a) Flow Table



(b) Pair Graph

Figure 5. Determination of ultimate distinguishability

All the pairs in Figure 5 are ultimately distinguishable except the state pair (1, 2). This criteria ensures the propagation of the effect of a fault to an observable output.

Another condition for testability is that it should be possible to apply all necessary inputs to any cell in the array (one-dimensional). Since Friedman has made a restricted fault assumption, the set of applicable inputs to an arbitrary cell need not contain all possible input states. Test derivation procedures involve, first, realization of the cell in a circuit form (according to the flow table), then the derivation of tests for faults (s-a-1, s-a-0) in each wire of the circuit, and finally the setting up of a table of faulty and normal outputs for each fault. Later a testing graph is constructed using the table of tests, for faulty and normal outputs as in

Figure 6.

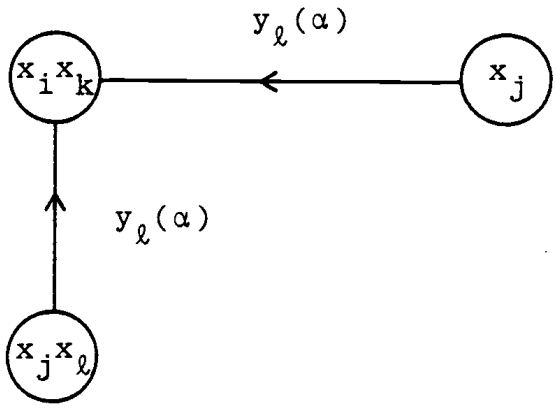


Figure 6. Testing Graph

If (x_j, y_ℓ) (y_ℓ is the y input and x_j is the input state) is the test for a fault $\alpha(\alpha\varepsilon\{\text{s-a-o, s-a-1}\})$ in the faulty cell then we draw an edge labelled y_ℓ (α) to (x_i, x_k) where x_i is the output corresponding to a normal transition from x_j on the input y_ℓ, and x_k is the faulty output (fault due to α). If there is another node (x_j, x_ℓ) which has an edge leading into (x_i, x_k) and labelled y_ℓ, then we append (α) to this label. This is so because the change $x_j \rightarrow x_\ell$ can be propagated using the input y_ℓ, and is equivalent to the fault 'α'.

After completing the graph an efficient fault detection test set is obtained by finding loops in the testing graph such that the edge labels in the loops cover the largest number of faults in the circuit. The external input label

corresponding to the edges in the loop form a test sequence (Figure 7).

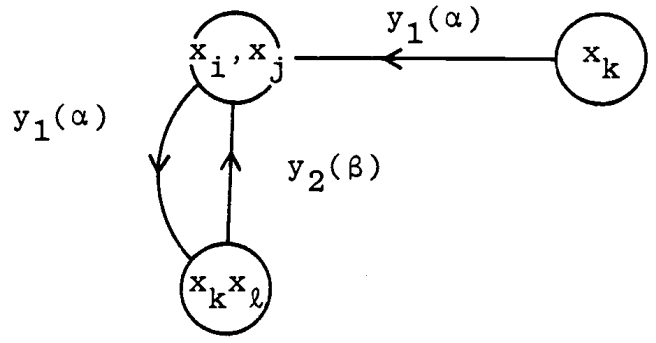


Figure 7. Testing graph

From the above partial testing graph we have two tests corresponding to the loop; i.e.,

(1) $x_i y_2 y_1 y_2 \cdots$

(2) $x_k y_1 y_2 y_1 \cdots$

These two tests will test for the faults 'α' and 'β' in a one-dimensional iterative array.

In order to locate the faulty cells in an array, use is made of masking inputs. A masking input (x_m, y_m) corresponding to a test t_i, $(t_i = (x_i, y_i))$ for a fault 'f_i' is one such that the output of the normal and faulty cells with the masking input applied is the same as the output of the normal cell with 't_i' applied to it; i.e., $\hat{x}_{f_i}(x_m, y_m) = \hat{x}(x_m, y_m) = \hat{x}(x_i, y_i)$.

Essentially the masking input masks the effect of the fault. A set of masking inputs is derived for all faults and if a particular test indicates a fault in the array, the corresponding masking input is applied to the first cell in the array. A correct external output indicates a fault in the first cell (since the masking input masks the fault in the first cell). If the external output were to be incorrect as before, the first cell will be fault free. We now apply the same masking input to the second cell as before and determine whether the second cell is faulty. In a similar manner we apply the masking inputs to all the cells in the array to locate the faulty cell. However masking inputs may not exist for all faults in a cell, hence the author extends the concept of masking input to blocks of cells (each block of two or more cells) resulting in location up to p-cell blocks ($p \geq 2$).

Turning to two-dimensional arrays, Friedman makes use of a diagonal tessellation where the cells along the diagonal of the array are all in the same state. This is called a $+45^{o}$ diagonal tessellation. As mentioned earlier, a successor graph is drawn and the states in a loop in that graph can be applied to all cells in the two-dimensional array. This is done by applying each input state in the loop to the leftmost corner cell in the array and the succeeding states in the loop to the successive diagonals in the array until the entire array is covered. However it

is not always possible to generate diagonal tessellations that cover all possible input combinations to a given array. But some inputs that were not applicable earlier may be applicable if we consider the tessellation of cell blocks containing more than one cell (Figure 8).

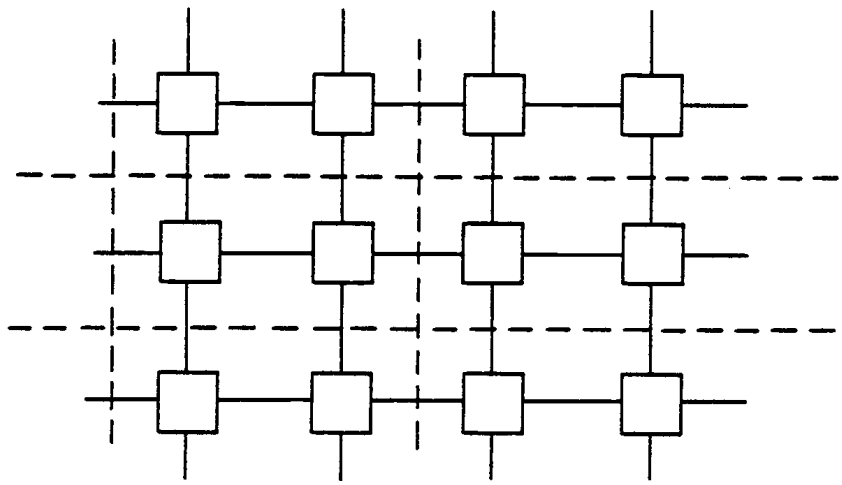


Figure 8. Diagonal tessellation of two-cell blocks.

A successor graph corresponding to Figure 8, where each node is of the form (x_1, y_1, y_2), is drawn and the input states $((x_1, y_1), (x_2, y_2))$ contained in a loop in the graph can be applied to all cells in the array. If there are some more input states that cannot be applied at this stage, a successor graph for three cell-blocks is drawn to determine whether these states can be applied to the array. The procedure gets cumbersome as the size of the cell block increases. Moreover this procedure may not

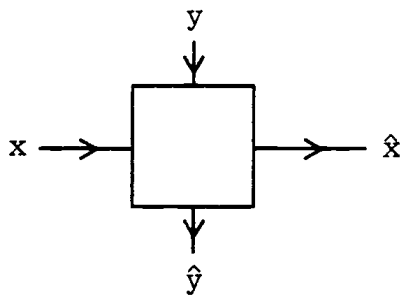
yield the complete set of applicable cell inputs and we have no method of determining when to terminate the procedure.

In essence the notion behind the use of tessellations is to generate a set of external inputs to the two-dimensional array such that all input combinations can be applied to all the cells in the array. Since each tessellation set corresponds to a loop in the successor graph, any faulty transition would result in an exit from the loop, indicating the presence of fault.

While Friedman uses a set of masking inputs to locate faulty cells within the array, Thurber [43] has evolved a test set for fault location in Maitra cascades (single faults) using a restricted fault set. Every cell in a Maitra cascade is a two-input, one-output cell. His approach involves the generation of a set of allowable errors for each cell type in the cascade and the use of a binary decision tree to locate the faulty cell.

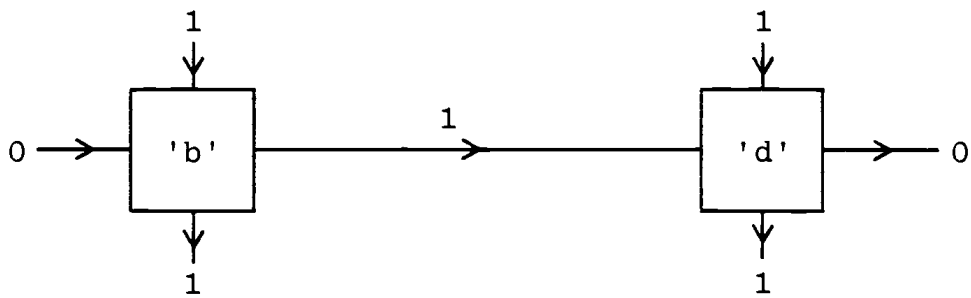
Returning to fault detection in the two-dimensional array, Chia and Coates [7] have introduced a procedure which enables one to tackle the problem of the existence of a tessellation. The set of tessellations is partitioned into composite and prime tessellations where the former corresponds to the set of input states contained in a loop in the successor graph and the latter is one which is not a composite tessellation. Chia defines each input-output

relation of a cell as the status of a cell.



For the above cell (x, y, $\hat{x}$, $\hat{y}$) is called the status of the cell. If $\hat{x} = \overline{x} + \overline{y}$ and $\hat{y} = \overline{x} + y$, the possible statuses are a = (0, 0, 1, 1), b = (0, 1, 1, 1), c = (1, 0, 1, 0), and d = (1, 1, 0, 1).

By assigning the status 'c' to a cell we can obtain a composite tessellation; i.e., an input combination corresponding to the status 'c' can be applied to all the cells in a two dimensional array. Similarly by assigning the status 'b' and the status 'd' to a 1 x 2 cell array we can achieve a composite tessellation.



We will need two tests to apply 'b' and 'd' to all the cells in the array, and one test to apply the status 'c'. However the status 'a' cannot be applied to all cells in an

array as mentioned above and hence is a prime tessellation. Thus an heuristic procedure is used to apply the status 'a' to all cells in the array and the number of tests required in the case depends on the dimension of the array. Figure 9 shows a portion of the prime tessellation in which the status is applied to the array. In order to apply 'a' to all cells in the array of size m x n we need (m + n-1) tests.

Since certain arrays are not easily testable, certain design modifications have been proposed so that the modified array can be tested easily. Seth [38] in his paper on the fault detection problem in two dimensional arrays has developed a set of desirable and undesirable function pairs - in terms of easier testability - where the latter should be avoided in the design of iterative arrays. The undesirable functions are those in which at least one input combination corresponding to a function cannot be applied to all cells in a two-dimensional array. For example, one input combination corresponding to the function pair ($\overline{xy}$, $\bar{x}$) cannot be applied to all the cells in a two-dimensional array. At least one of the functions in an undesirable function pair is a nodal function; i.e., a nodal function is one which contains either a single 0 or a single 1 in the truth table representation of the function. If f is a nodal function, there is only one input combination which is mapped to the constant value

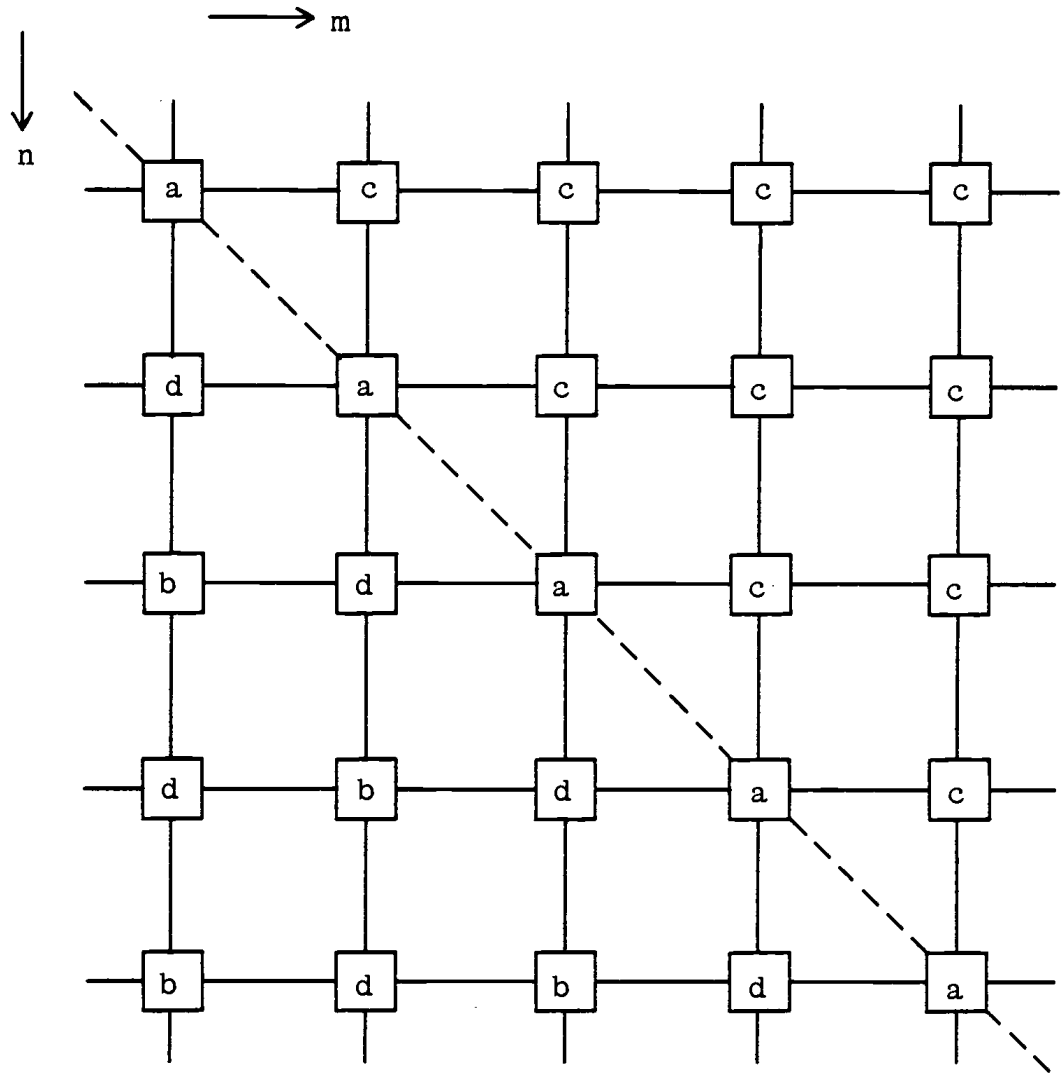


Figure 9. An example of prime tessellation.

c(0 or 1) and all other combinations are mapped to $\bar{c}$ under f.

Later, Landgraff and Yau [21] proposed more conditions in addition to those proposed by Kautz [17] in connection with the testability of one and two dimensional arrays. The additional conditions were needed for flow tables that did not satisfy Kautz' conditions. A suitable procedure is stated for the modification of the flow tables that do not satisfy the conditions for testability, thereby making them easily testable. Basically this procedure identifies certain types of transitions in the flow graph of the cell, say, for example, self loops, and discusses their correlation to easier diagnosability of the array.

Yet another design modification was suggested by Prasad and Gray [35]. Under an assumption of multiple, unrestricted faults they have developed tests for flow tables that have one permutation column. If the input corresponding to the permutation column in the flow table is applied to the external input of all the cells in the array; i.e., cells to the right of the faulty cell, the effect of any fault is guaranteed to propagate to the end of the array. The same results are extended to two-dimensional arrays also.

Recently Dias [9] developed another procedure to determine a test set for fault detection in a one-dimensional array. Using the concept of distinguishing

sequence (an input sequence whose application to a flow table produces different output sequences for each choice of the initial state) he derives tests for checking each state transition in the given flow table. The test sequence derived is cyclical in nature and hence checks simultaneously more than one cell for the corresponding state transition. The test set thus derived is sufficient to verify the truth tables of the normal and the faulty array, thereby enabling multiple-fault detection in the array.

The discussion thus far gives some idea of the different approaches taken in the past for generating test sets capable of detecting faults in one and two-dimensional arrays. Insofar as multiple-cell failures are concerned, only Prasad and Dias have developed techniques to detect them, the latter only in case of one-dimensional array. Prasad has concerned himself with a specific flow table and the easier test generation procedure with respect to such flow tables. The number of tests in his case depends on the number of cells in the array.

Hence we will turn our attention in this dissertation to the development of an algorithm for generating test sets that are capable of detecting multiple-cell faults. This will be independent of the nature of individual cell design as well as the number of cells in the array. Multiple-cell faults are more prevalent in LSI technology

than single cell failures. While Kautz [17] has addressed himself to the problem of fault detection in flow tables with permutation columns, he has not extended his techniques to an arbitrary flow table that possesses no permutation columns.

The algorithm derived in this dissertation will generate tests for flow tables that obeys a graph theoretic condition which is less restrictive than earlier methods. The tests derived are generally such that each will simultaneously test more than one cell in the array for the given fault. In the case of Dias [9] the loop tests require an external output corresponding to each cell in the array. Tests generated by our algorithm do not require any external outputs and the presence of such output in each cell will in no way affect the generation of the test set.

In case of two-dimensional arrays, earlier work has mostly been confined to the determination of tessellation sets that cover all possible input combinations to the array. The approach we have taken differs considerably from the above approach in that the two-dimensional array is viewed as an equivalent one-dimensional array. This is achieved by compressing the two-dimensional array either horizontally or vertically. Such an approach facilitates the extension of the results derived for the one-dimensional case to the two-dimensional case too. As in the one-

dimensional case, tests cyclical in nature do not exist for all faults in a two-dimensional array. Hence it is seen from the above that our approach is new compared to the earlier work and that it addresses itself to the problem of multiple-cell-failures, which is a more realistic assumption in the context of contemporary technology.

CHAPTER III

MULTIPLE FAULT DETECTION IN THE ONE-DIMENSIONAL ITERATIVE ARRAY

3.1: Introduction

Over the past decade developments in integrated circuits have turned the attention of designers toward the structural simplicity and design of systems using a standard set of subcircuits. Such structural simplicity is found in a class of circuits known as iterative circuits, that can be either combinational or sequential in nature. Identical cells are connected in a one or two-dimensional pattern, the connections being mere conductive paths without any logic. Configurations such as these result in a higher packing density, low cost, high functional performance and simplified fault diagnosis, (Minnick [31], Kautz [18, 19], Jump [16]). There are also arrays that have been designed with different types of interconnections, (Short [41], Giovene [14], Akers [2], Maruoka [24]). However the testing of large arrays, in spite of their structural elegance, is still a complex task.

In this dissertation, a combinational iterative array is an array of identical combinational cells connected at intercell edges either in a one or two-dimensional rectangular pattern. The signals are assumed to flow from left to right in the former case and top to bottom as well in

the latter. These are known as unilateral intercell connections, Hennie [15]. Figures 10 and 11 show a one- and two-dimensional iterative combinational array respectively. Each cell in Figures 10 and 11 is identical. The internal logic is assumed to be combinational in nature. X_i is the horizontal input and is a vector, i.e., $X_i = \{x_i^1, x_i^2, \ldots x_1^n\}$. Similarly Y_i is the vertical input and is a vector, i.e., $Y_i = \{y_i^1, y_i^2, \ldots y_i^n\}$. $\hat{X}_i$ and $\hat{Y}_i$ are the corresponding outputs. An analogy can be made between iterative and synchronous sequential circuits, (McCluskey [25], Hennie [15]) thus enabling us to view the X_i's and $\hat{X}_i$'s as input and output states and the Y_i, $\hat{Y}_i$'s as external inputs and outputs. This view allows us to represent the terminal behavior of a cell by a flow table of dimension (n x m) where 'n' is the number of rows in the flow table and 'm' is the number of columns. The entries in the flow table represent the output state and the external output corresponding to that row and colunn. The behavior of the array is determined by the mapping $X_i \times Y_i \longrightarrow \hat{X}_i \times \hat{Y}_i$. (i=1 in the case of one-dimensional array.)

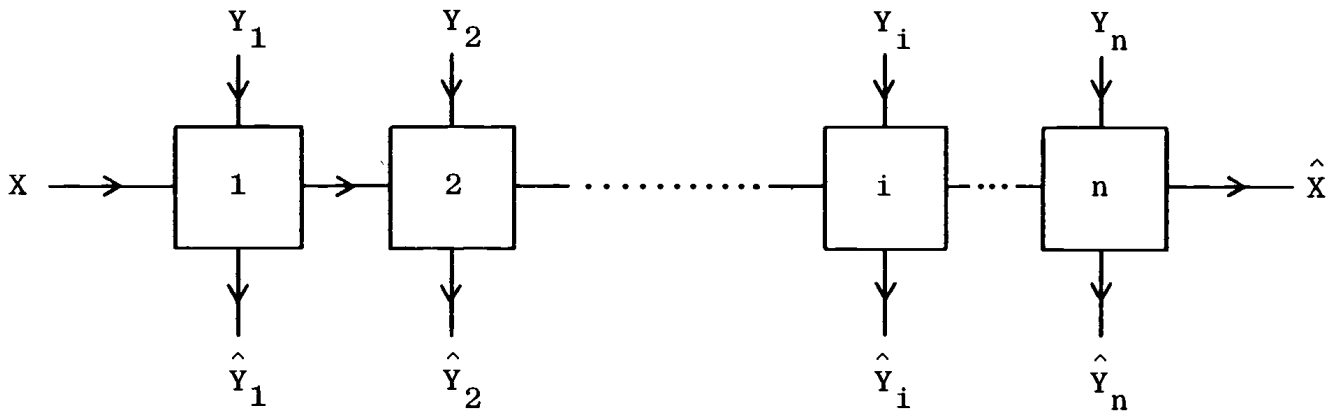


Figure 10. One-Dimensional Array

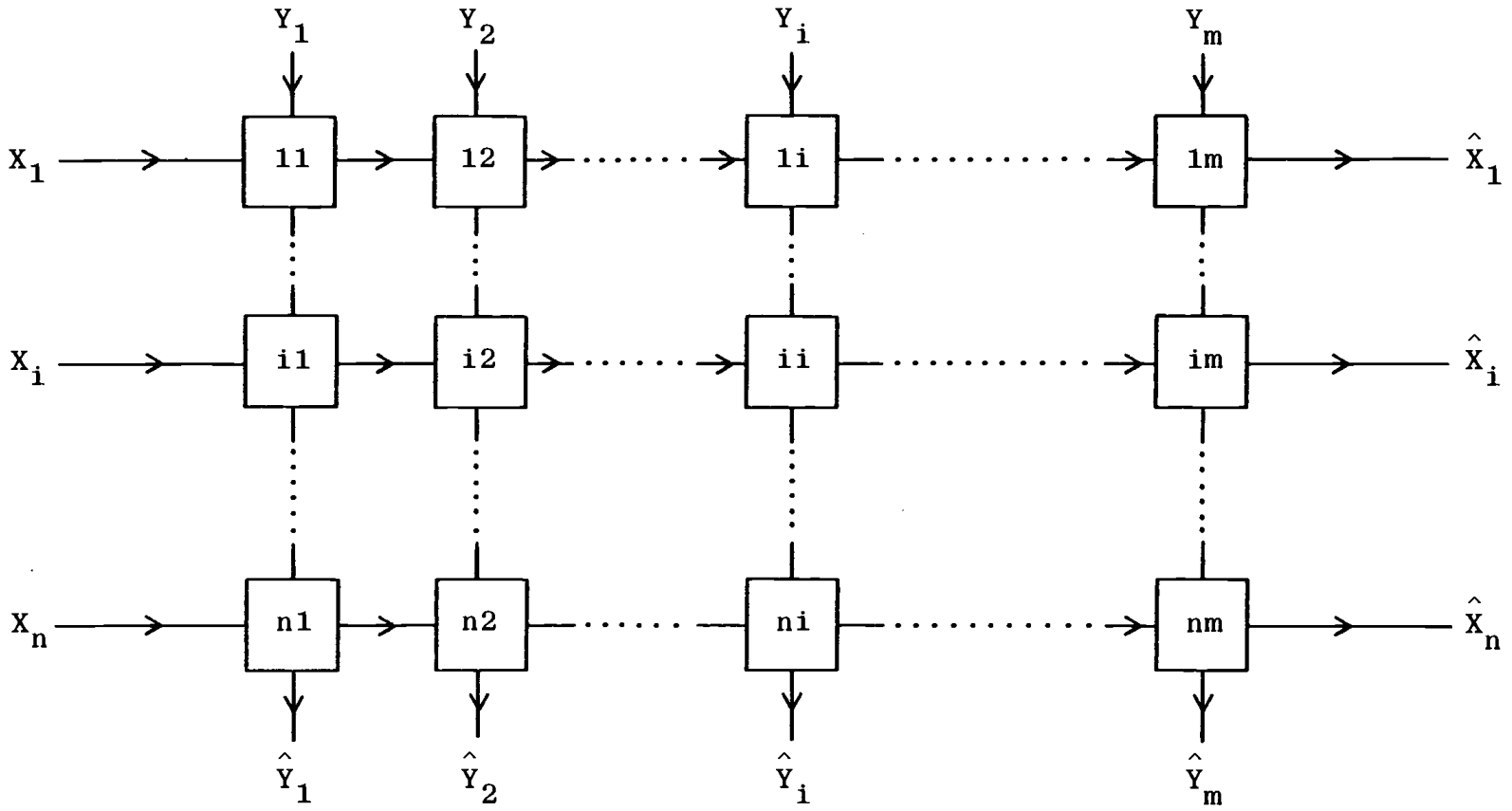


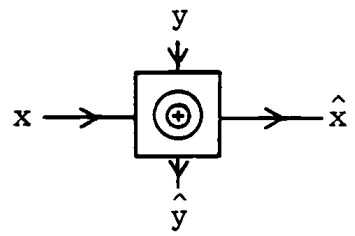
Figure 11. Two-dimensional array.

Example: If each cell in a one-dimensional array is a realization of an Ex-OR function the corresponding flow table for a cell in the array is as follows.

Flow Table:

x \ y	0	1
0	0	1
1	1	0

Cell Realization:



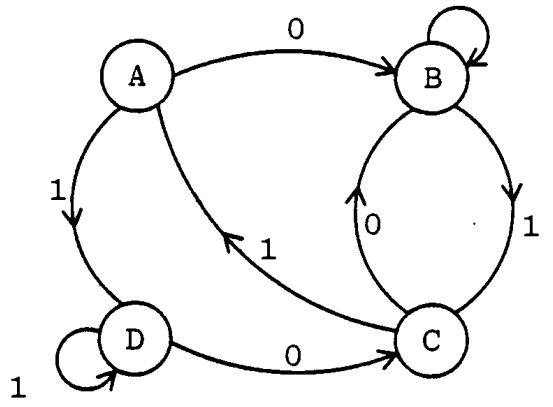
3.2: Assumptions

Before proceeding further let us state the basic assumptions on the nature of faults in which we are interested. The assumptions are:

1. All faults occurring in the array are permanent; the fault remains until it is corrected.
2. The individual cell should be tested for all possible faults by applying all possible inputs to the cell.
3. Due to the second assumption each state should appear at least once in the flow table and the corresponding state graph should be strongly connected and reduced.

Example:

	0	1
A	B	D
B	C	B
C	B	A
D	C	D



The above state graph corresponding to the flow table is strongly connected in that any state can be reached from any other state and each state has appeared at least once in the flow table.

4. Faults may occur within each cell as well as on the intercell leads.

Fault analysis in digital circuits has generally been directed toward s-a-0 and s-a-1 faults, [2]. In our case we will be interested in a broader class of faults known as state faults, [17]. An individual cell may have single or multiple faults within the cell but each of these faults results in a faulty transformation to a state other than the correct one. Each such faulty transformation will result in a new flow table, one different from the normal flow table. Such faults are defined as State Faults. In effect, any other component failure that does not cause a state fault is not recognized as a fault, and is acutally masked by the circuit design itself. In

order to test for all possible state faults it is necessary that we should be able to apply all possible input combinations to each cell in the array.

At this point we will define propagation of faults. Assigning suitable inputs to an iterative array such that the array output state (external array output) is different under faulty and normal conditions is defined as propagating the effect of the corresponding fault. Propagation of state faults requires that there exists a sequence of external inputs capable of transmitting the effect of such faults to an external output where it is observable. The following lemma will help us prove the existence of such a sequence for a reduced flow table.

LEMMA 3.1: An input sequence that distinguishes between any two states in a flow table exists if the flow table is reduced.

Proof: Let $S_1,\ldots,S_n$ be the states in the flow table, and $Y_1, Y_2,\ldots Y_m$ represent the external inputs. Let S_i and S_j be the two states that are to be distinguished.

Since the flow table is reduced there always exists an input $Y_{k1}(ki = 1,\ldots,m)$ that distinguishes S_i and S_j, i.e.,

$S_i \longrightarrow \hat{S}_i$

$S_j \longrightarrow \hat{S}_j$

($\hat{S}_i \neq \hat{S}_j$ but $\hat{S}_i$ can be equal to S_i and $\hat{S}_j$ can be equal to S_j)

Similarly $\hat{S}_i$ and $\hat{S}_j$ can be distinguished by another input Y_{k2}. Proceeding in this fashion we can arrive at a sequence of inputs $Y_{k1}Y_{k2}...Y_{km}$ ($m \leq n-1$) that distinguishes S_i and S_j. Hence all faulty state transformations can be propagated to an external output and so are testable.

3.3.: Fault Model

In order to analyze the one dimensional iterative array for its fault behavior consider an array modelled as in Figures 12 and 13. The iterative array can be viewed as partitioned into three modules. Initially we assume a singly faulty cell and later will extend to the case of multiple cell fault condition. In the above model the faulty cell is isolated as an individual module, the fault cell (FC). Cells that precede it are grouped together as the fault free array (FFA), and the group of cells that succeed it as the fault propagation array (FPA). Since we have assumed unilateral signal flow, the FFA module is independent of the FC and the FPA module in the iterative array. The output function of the FFA can be written as $\hat{f}_{FFA}$ where $\hat{f}_{FFA} = \hat{f}\ (X, Y_j\ ;\ j=1,...,i-1)$ and recursively the output of the faulty cell is $\hat{f}_{FC} = f(\hat{f}_{FFA}, Y_i)$. The output of the FPA is $\hat{f}_{FPA} = \hat{f}(\hat{f}_{FC}, Y_k\ ;\ k=(i+1),...,n)$.

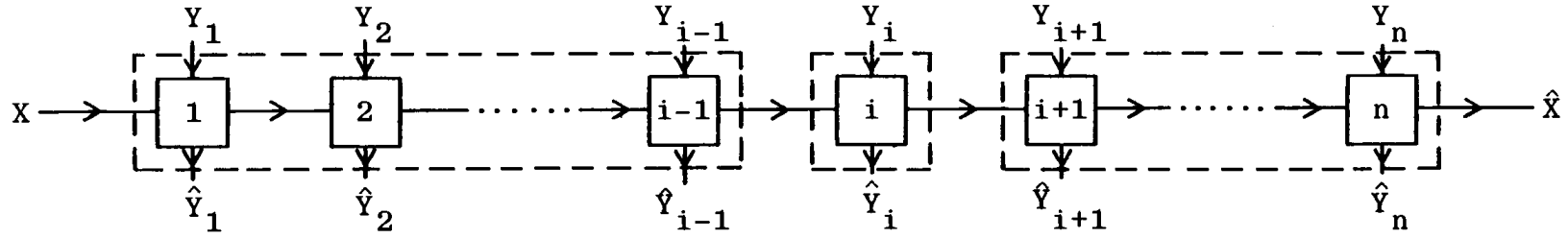
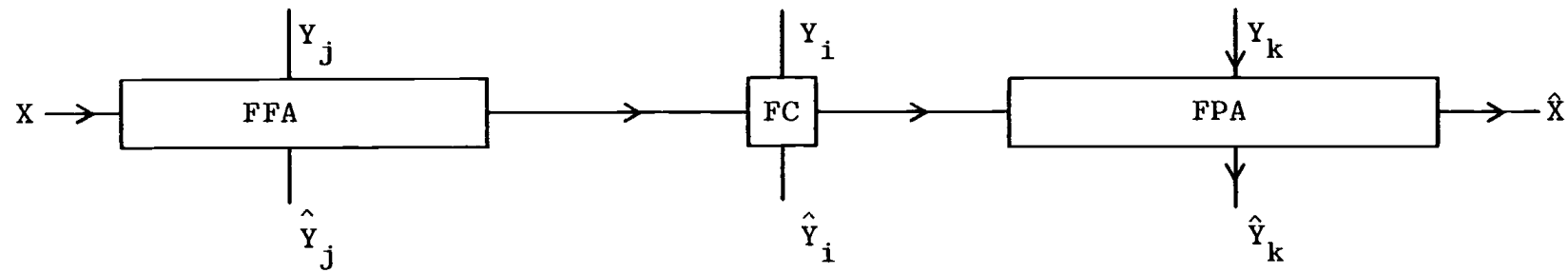


Figure 12. One-dimensional iterative array with a faulty cell.



((j=1,...,i-1),(k=i+1,...n))

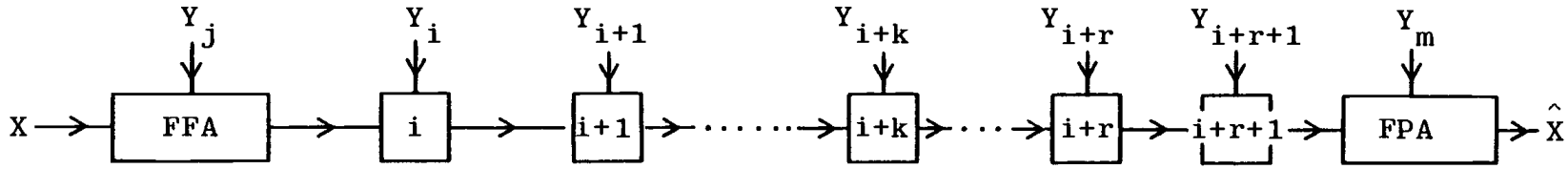
Figure 13. One-dimensional array partitioned into faulty and non-faulty modules.

The $\hat{f}$'s are functional equivalents to the output states of each module. The external outputs are considered as additional appendages and in our analysis they are not taken into consideration (according to our initial assumption). A test for a detectable fault in a faulty cell (FC) would correspond to a sequence 't' where 't' is a function of X and Y, i.e., $t=(X, Y_1, Y_2,\ldots,Y_{i-1}, Y_i, Y_{i+1},\ldots,Y_n)$, $X, Y\varepsilon(0,1)$.

Turning our attention to multiple cell faults, we now prove that tests derived under a single fault assumption will also be able to detect multiple cell faults.

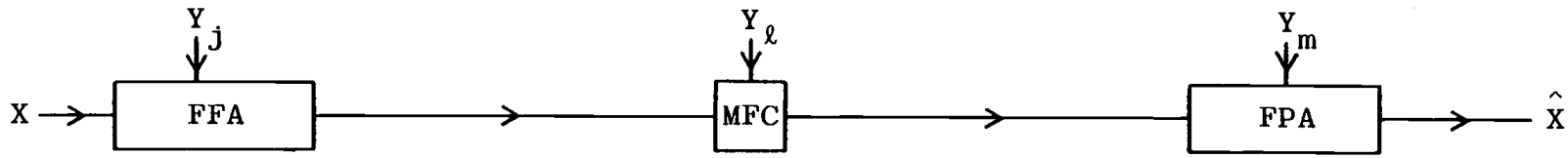
THEOREM 3.1: Multicell faults (multiple-faults) in a one-dimensional iterative combinational array can be detected using the same tests derived under the single fault assumption.

Proof: The model corresponding to single cell fault given in Figure 14 is modified in Figure 15 to represent a multiple cell fault condition. If we assume an arbitrary number of cells to be faulty, say the i, $i+k_1$, $i+k_2,\ldots i+k_j$ cells to be faulty, then Figure 14 can be redrawn as in Figure 15. Let two cells i and i+r be faulty in the given one-dimensional iterative array. Then in Figure 15 $\ell = i, (i+1),\ldots,(i+r)$. Assume the fault on i^{th} cell to be a faulty state transformation $S_m \xrightarrow{I_\ell} S_q$ instead



$(m = (i+r+2), \ldots, n)$

Figure 14. Iterative array with k faulty cells.



$(j=1, 2, \ldots(i-1);\ \ell=i, \ldots, i+k_j; m=i+k_j+1, \ldots n)$

Figure 15. Model for multiple fault cell array.

of $S_m \xrightarrow{I_\ell} S_p$ which is the fault-free transformation. Similarly let the fault in the $(i+r)^{th}$ cell be $S_a \xrightarrow{I_c} S_d$ where $S_a \xrightarrow{I_c} S_b$ is the fault-free transformation. Let the tests derived for the i^{th} and $(i+r)^{th}$ cell faults under the single fault assumption be t^q_{mp} and t^d_{ab}.

i.e., $t^q_{mp}=(x_m, y_{1m},\ldots y_{(i-1)m}, I_1,\ldots, y_{nm}; x_m \varepsilon X, y_{im} \varepsilon Y_i)$

$$t^d_{ab}=(x_a, y_{1a},\ldots y_{(i+r-1)a}, I_c,\ldots, y_{na}; x_a \varepsilon X, y_{ia} \varepsilon Y_i)$$

Let us now assume both the i and $(i+r)^{th}$ cells to be faulty at the same time (multicell fault), and the faults are as mentioned above.

Case 1. In Figure 16 let $S_{(i+r)} \xrightarrow{y_{(i+r)m}} S_{(i+r)}$ be the input-output state transition of the $(i+r^{th}$ cell) on an external input $y_{(i+r)m} (y_{(i+r)m} \varepsilon t^q_{mp})$. Let $\hat{S}_{nm}(N)$ be the output state of the n^{th} cell when both i and $(i+r)^{th}$ cell are fault free and $\hat{S}_{nm}(F)$ be the output state when only the i^{th} cell is faulty. $\hat{S}_{nm}(N)$ and $\hat{S}_{nm}(F)$ are responses to the test sequence t^q_{mp} under the two conditions described above, $(\hat{S}_{nm}(N) \neq \hat{S}_{nm}(F))$. The propagation of the fault in the i^{th} cell under the presence of a fault in the $(i+r)^{th}$ cell depends on the nature of $S_{(i+r)}$.

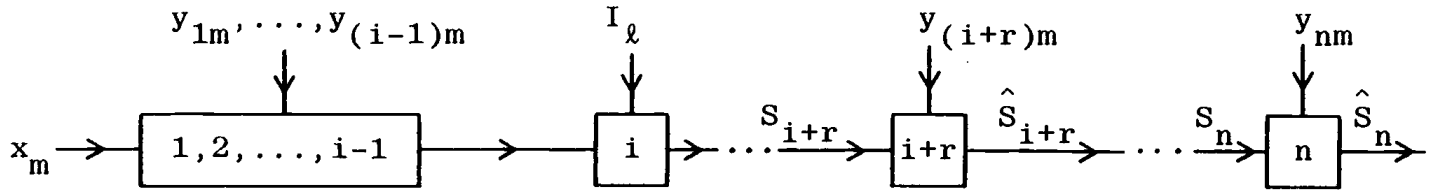


Figure 16. Application of t^q_{mp} to the one-dimensional array.

(a). $S_{(i+r)} \neq S_a$

In this case the fault in the $(i+r)^{th}$ cell, i.e., $S_a \xrightarrow{I_c} S_d$, will not affect the propagation of the i^{th} cell fault. Hence application of t^q_{mp} will detect the presence of the i^{th} cell fault in the array.

(b). $S_{(i+r)} = S_a$ and $y_{(i+r)m} = I_c$

Because of the above relation we now have the following.

$$S_{(i+r)} \xrightarrow{y_{(i+r)m}} S_{(i+r)} = S_a \xrightarrow{I_c} S_d$$

$$S_d \xrightarrow{y_{(i+r+1)m}, \cdots, y_{nm}} S_e$$

Corresponding to S_e we have two possibilities, (S_e is the output state of the n^{th} cell).

1. $S_e = \hat{S}_{nm}(N)$, i.e., if the output S_e is the same as the one under normal condition, the effect of the $(i+r)^{th}$ cell fault has cancelled out the effect of the i^{th} cell fault. In essence these two faults have masked each other. Hence the fault will not affect the normal functioning of the array.

2. $S_e=\hat{S}_{nm}(F)$. Here the output S_e is the same as the one under faulty condition and hence the i^{th} cell fault is detected under the presence of the (i+r)th cell fault.

Case 2. In Figure 17 let $S_i \xrightarrow{y_{ia}} \hat{S}_i$ be the input-output state transition of the i^{th} cell on an external input $y_{ia}(y_{ia} \varepsilon t^d_{ab})$. $\hat{S}_{na}(N)$ and $\hat{S}_{na}(F)$ are the n^{th} cell outputs (response to t^d_{ab}) on normal (both i^{th} and $(i+r)^{th}$ cell fault free) and faulty ($(i+r)^{th}$ cell faulty) condition, $(\hat{S}_{na}(N) \neq \hat{S}_{na}(F))$.

The question whether the input state S_a can be applied to the $(i+r)^{th}$ cell under the presence of the fault in the i^{th} cell depends on the nature of S_i.

(a). $S_i \neq S_m$

In this case the fault in the i^{th} cell will not affect the value of S_{i+r} and S_{i+r} will be equal to S_a. Hence the test t^d_{ab} will detect the fault in the $(i+r)^{th}$ cell under the presence of fault in the i^{th} cell.

(b). $S_i=S_m$ and $y_{ia}=I_1$

The above relation leads to the following transitions.

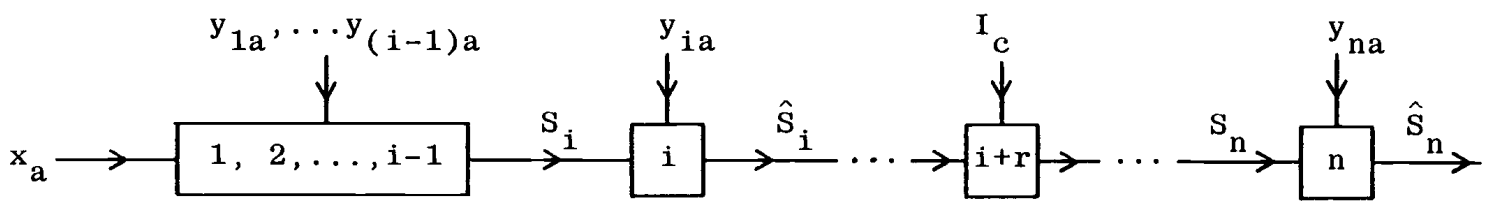


Figure 17. Application of t_{ab}^{d} to the one-dimensional array.

$$S_i \xrightarrow{y_{ia}} \hat{S}_i \quad = \quad S_m \xrightarrow{I} S_q$$

$$S_q \xrightarrow{y_{(i+1)a}, \ldots, y_{na}} S_o$$

Corresponding to S_o we have two possibilities as before.

1. $S_o = \hat{S}_{na}(N)$. i.e., the output S_o is the same as the one under normal condition. Hence the effect of i^{th} cell fault is to cancel out the effect of $(i+r)^{th}$ cell fault. The two faults have masked each other.
2. $S_o = \hat{S}_{na}(F)$. Here the output S_o is the same as the one under faulty condition, hence the $(i+r)^{th}$ cell fault is detected under the presence of i^{th} cell fault.

We conclude from both case (1) and (2) that the faults in both cells (multiple cell faults) can be detected by t^q_{mp} and t^d_{ab}, each generated under single-fault assumption. The above result can be easily extended for k-cell faults, k>2. In the above theorem we haven't mentioned multiple faults within the same cell. This is due to the fact that both single as well as multiple faults within the cell result in erroneous state transitions in most cases. Such erroneous state transitions are already included in our state fault assumptions.

3.4: Some Useful Graph Theoretic Tools

In our analysis of fault detection in iterative arrays we shall make use of certain basic techniques from graph theory. Let us agree on certain basic definitions [8] before proceeding with the actual test generating procedure.

Each cell in an iterative array is identical except in the case of certain special arrays such as Minnick's cut-point cellular array where each cell realizes a different function. A cell in an iterative array is represented by a state graph or a flow table. The state graph is known as a digraph. A digraph (directed graph) G consists of a set of vertices $V = (v_1, v_2, \ldots, v_n)$ and a set of edges $E = (e_1, e_2, \ldots)$. Every edge is associated with an ordered pair of vertices (v_i, v_j). A labelled digraph can be interpreted as a state graph, where the vertices correspond to the set of states in the flow table and the edges to transitions corresponding to the external inputs.

A walk in a digraph is defined as a finite, alternating sequence of vertices and edges such that each vertex preceeding an edge is incident on the vertices succeeding the edge, but no edge appears more than once. A closed walk is one that starts and ends at the same vertex. A path is an open walk (a walk that is not closed) in which no vertex appears more than once. In a digraph such a path

is known as a directed path. A digraph is said to be strongly connected if there is at least one directed path from every vertex to every other vertex. A closed walk in which no vertex (except initial and final vertex) appears more than once is called a circuit or cycle.

In order to manipulate graph structures, matrices can be used as a way of representation and are often found to be easy and convenient to operate upon. A state graph can be represented as a transition matrix [39] T where the rows and columns correspond to initial and final states. Each entry e_{ij} in the matrix corresponds to the label ℓ_k on the edge connecting state S_i and S_j. i.e., $S_i \xrightarrow{\ell_k} S_j$. In the event of parallel edges (more than one edge connecting the same ordered pair of states) the entry e_{ij} can be written as $(\ell_{k1} + \ell_{k2} + \ldots)$ where ℓ_{k1}, ℓ_{k2} are labels on the edges connecting S_i and S_j and '+' correspond to the OR function, denoting the fact that transitions from S_i can take place either on ℓ_{k1}, or ℓ_{k2}, etc. If there is no edge corresponding to a pair of states, then, such an entry is marked as $\emptyset$. The number of non - $\emptyset$ entries in the i^{th} row of the matrix corresponds to the out-degree of the vertex S_i. In essence the T matrix represents the state transitions of a typical cell in the iterative array.

T-Matrix Multiplication:

Definition: T^i is a matrix of power i., i.e.,

$T^i = TxTxTx...T.$ (i times)

Definition: $(e_{ij})^k$ represents an entry in the T^k matrix and is a sequence of length k that transforms S_i into S_j.

$T^i \times T^j$ is commutative, i.e., $T^i x T^j = T^j x T^i$

T-Matrix multiplication follows the same rule as any other matrix except for the following:

Rule 1: If two entries $(a_{ij})^r$ and $(b_{jk})^s$, corresponding to T^r and T^s, are multiplied, the resulting entry $(c_{ik})^{r+s}$ in T^{r+s} will be $(a_{ij})^r (b_{jk})^s$. In other words $(b_{jk})^s$ is just appended to $(a_{ij})^r$. $(c_{ik})^{r+s}$ is a sequence of length (r+s) in the T^{r+s} matrix that transforms S_i into S_k.

Rule 2: If $(a_{ij})^r = \emptyset$ then $(a_{ij})^r (b_{jk})^s = (c_{ik})^{r+s} = \emptyset$

From the above two rules we see that entries in a T^k matrix correspond to sequences of length k, and that $(e_{ij})^k$, an entry in the T^k matrix, represents the sequence of inputs $(y_i, y_{i+1} \ldots y_k)$ that will transform the state S_i

in the first cell (in a one-dimensional iterative array) to S_j at the output of the k^{th} cell. The following examples illustrate the representation of a flow table by means of a transition matrix and multiplication of such matrices.

Example 1: To illustrate Rule 1:

Flow Table:

x \ y	0	1
A	A	B
B	B	A

T-Matrix Representation:

$$T = \begin{array}{c|cc} & A & B \\ A & 0 & 1 \\ B & 1 & 0 \end{array} \quad \text{where } e_{AA}=0,\ e_{AB}=1,\ e_{BA}=1,\ e_{BB}=0$$

T-Matrix Multiplication:

$$TxT = T^2 = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} X \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} =$$

$$= \begin{vmatrix} (0)(0)+(1)(1) & (0)(1)+(1)(0) \\ (1)(0)+(0)(1) & (1)(1)+(0)(0) \end{vmatrix}$$

$$= \begin{array}{c|cc} & A & B \\ A & 00+11 & 01+10 \\ B & 10+01 & 11+00 \end{array}$$

where $(e_{AB})^2=\{(e_{AA})(e_{AB}),(e_{AB})(e_{BB})\}$

In T^2, $(e_{AB})^2 = \{01,\ 10\}$

i.e., 00 and 11 are the external y inputs that would take state A back to itself.

Example 2: To illustrate Rule 2:

Flow Table:

x \ y	0	1
A	A	A
B	B	A

T-Matrix Representation:

$$T = \begin{array}{c|cc} & A & B \\ A & 0+1 & \emptyset \\ B & 1 & 0 \end{array}$$

$$T^2 = \begin{vmatrix} 0+1 & 0 \\ 1 & 0 \end{vmatrix} \quad \begin{vmatrix} 0+1 & \emptyset \\ 1 & 0 \end{vmatrix}$$

$$= \begin{vmatrix} (0+1)(0+1)+(\emptyset)(1) & (0+1)(\emptyset)+(\emptyset)(0) \\ (1)(0+1)+(0)(1) & (1)(\emptyset)+(0)(0) \end{vmatrix}$$

$$= \begin{vmatrix} 00+01+10+11 & \emptyset \\ 10+11+01 & 00 \end{vmatrix}$$

For simplicity the '+' signs in the T^2 matrix can be omitted and instead ',' is substituted. Hence,

$$T^2 = \begin{vmatrix} 00,01,10,11 & \emptyset \\ 10,11,01 & 00 \end{vmatrix}$$

In the above T^2 matrix since $(e^1_{AB})^2$ is a $\emptyset$ entry, there is no sequence of length two that will transform state A to state B. It should be noted here that the

entries in the i^{th} row of the T^k matrix represent all possible external y input sequences of length 'k'. Since we have assumed a strongly connected graph, each column of a given T^k matrix should have at least one non-$\emptyset$ entry. If all the entries in a column of the T^k matrix are $\emptyset$ entries then the state graph is not strongly connected with respect to the state in that column.

3.5: Analysis of Concurrent Testing Procedure

Let us assume there are 'n' states in a flow table representation of a cell in the iterative array. Corresponding to every fault-free state transition there can be (n-1) faulty transitions. Hence the number of faults to be detected are (mn(n-1)), where 'm' is the number of external inputs in the flow table. i.e., the number of columns in the flow table. In order to be able to detect all the faults in a p-cell one-dimensional iterative array we should be able to apply all possible input combinations to all cells in the array. Such a process, without any simplification, would involve the application of (mn(n-1)p) tests in order to completely test the array. The number of tests required will be far less if a test derived for a particular faulty transition can test simultaneously every ℓ^{th} cell, (ℓ<p), where 'ℓ' is the length of the test. However such test sequences do not necessarily exist for all

faulty transitions corresponding to a flow table. Later in this section a condition for the existence of such test sequences is stated as a theorem.

It is the intent of this dissertation to develop an algorithm that will generate test sequences of the above nature using the transition matrix. Such a test will be called Concurrent Test and denoted as a C-Test. As stated earlier a fault in a cell is taken to be a faulty state transition. If $S_i \xrightarrow{I_d} S_j$ is the fault-free transition according to the given flow table, where I_d is the external y input, then $S_i \xrightarrow{I_d} S_k$, $(\forall\ k \neq j)$, is a faulty transition corresponding to the transition of S_i on an input of I_d.

Let the i^{th} cell be faulty in the given one-dimensional array and the faulty transition in that cell be $S_i \xrightarrow{I_d} S_k$. In order to detect the presence of the above fault, $(y_{i+1},\ldots,y_p) \varepsilon t_{ij}^k$, when applied to the FPA part of the array, should propagate the effect of the fault to the p^{th} cell output where it is observable. That is, the output state at the p^{th} cell due to $(y_{i+1},\ldots,y_p)$ should be different for the faulty array as opposed to the fault-free array.

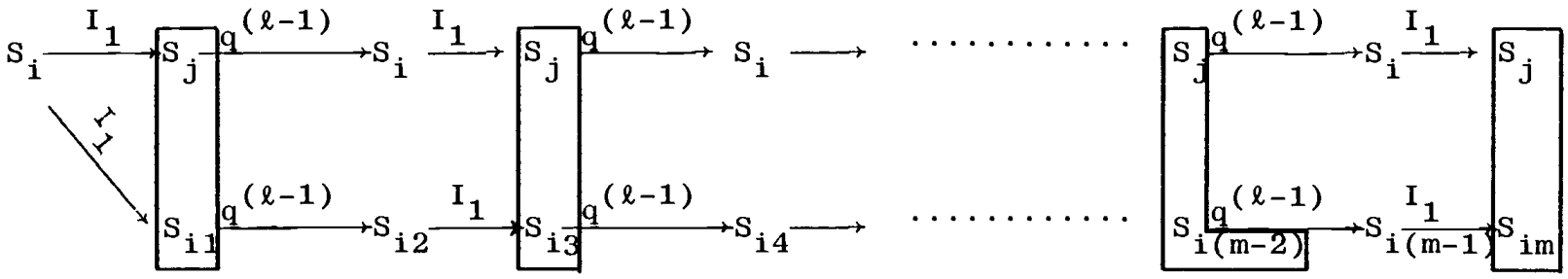
Thus,

$$S_i \xrightarrow{I_d} S_j \xrightarrow{y_{i+1},\ldots,y_p} S_r$$
$$S_i \xrightarrow{I_d} S_k \xrightarrow{y_{i+1},\ldots y_p} S_t \qquad (S_r \neq S_t)$$

where $(y_{i+1},\ldots,y_p)$ is of length $(p-(i+1))$. From lemma 3.1 we know that such a sequence exists for any given pair of states in the flow table. The upper bound on the length of the sequence will be established later in this chapter. Testing the entire array for the above state fault would presumably require p tests, one for each cell. However we will reduce the number of tests by using C-Tests that are also capable of detecting all detectable multicell faults.

Given the flow table of n states, let $S_i \xrightarrow{I_1} S_j$ be the fault-free transition and $S_i \xrightarrow{I_1} S_{i1}$ be the faulty transition. Assume the i^{th} cell to be faulty. Let us analyze the nature of a C-Test with the help of the following figure, Figure 18.

In Figure 18 each marked rectangle (S_j, S_{im}) represents a fault-free state (at the top of the rectangle) and faulty state (at the bottom of the rectangle) transition from S_i on I_1. Such a pair will henceforth be called as normal-and-faulty-state-pair (NFP). From Figure 18 we



$im \in (1,2,\ldots,n)$; $im \neq j \; \forall$ m corresponding to the pair $\boxed{\begin{matrix} S_j \\ S_{im} \end{matrix}}$

$im \neq i \; \forall$ m corresponding to the pair $\boxed{\begin{matrix} S_i \\ S_{im} \end{matrix}}$

Figure 18. State transitions when $q^{\ell-1}$ is applied to S_j and S_{i1}

define ($I_1\ q^{\ell-1}$) as a C-Test that will detect the presence of the faulty transitions from S_i on an input I_1 if (ir=ik) for r=3,5,...,m; k=1,3,5,...,m, i.e., if the state pairs in the enclosed rectangle (NFP) repeat. The C-Test will apply to every ℓ^{th} cell ('ℓ' is the length of the C-Test ($I_1\ q^{\ell-1}$)) in the array, the input combination (S_i, I_1), and will be able to detect the faulty transition to S_{im} in these cells. The number of tests needed to test for a particular fault depends on 'ir' and 'ik'. i.e., if r=3 and k=1 in Figure 18, then $S_{i3}=S_{i1}$. The following figure, Figure 19 illustrates this transition.

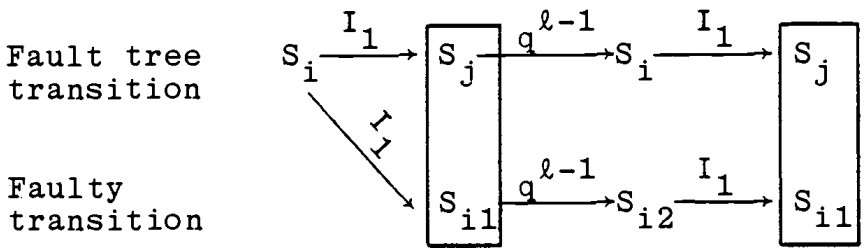


Figure 19. State transitions from S_j and S_{i1} when $S_{i3} = S_{i1}$ in Figure 18.

According to Figure 19 the C-Test will be ($I_1 q^{\ell-1}$) of length 'ℓ' and we will need 'ℓ' tests to test the entire array for the faulty transition $S_i \xrightarrow{I_1} S_{i1}$. On the other hand if r=5 and k=1, $S_{i5}=S_{i1}$ in Figure 18. Once again we will use another figure, Figure 20 to show this transition.

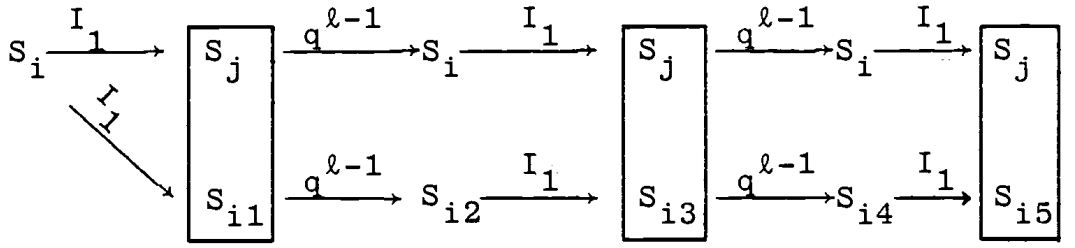


Figure 20. State transitions from S_j and S_{i1} when $S_{i5}=S_{i1}$ in Figure 18.

In the above figure ($I_1\ q^{\ell-1}$) will be the C-Test and we will need (2ℓ) tests to test the entire array for the faulty transitions $S_i \xrightarrow{I_1} S_{i1}$ and $S_i \xrightarrow{I_1} S_{i3}$. Similarly for r=m and k=1 we will need $m\ell$ tests to test the entire array for the faulty transitions $S_i \xrightarrow{I_1} S_{i1}$, $S_i \xrightarrow{I_1} S_{i3}, \ldots, S_i \xrightarrow{I_1} S_{i(m-2)}$. Later in the chapter we discuss two cases when $\ell \leq p$ and $\ell > p$ where 'p' is the number of cells in the array.

At this point we will illustrate the generation of a C-Test with an example.

Example: Consider the following flow table representing a typical cell in the array.

x \ y	0	1
A	B	A
B	C	D
C	D	B
D	C	A

Let us assume the faulty transition to be $A \xrightarrow{1} B$ as opposed to the fault-free transition $A \xrightarrow{1} A$. In order to detect this fault the following transition similar to Figure 18 will give us the C-Test. (A method to generate such a test is the topic of next section. Hence we assume here that the C-Test has already been generated.)

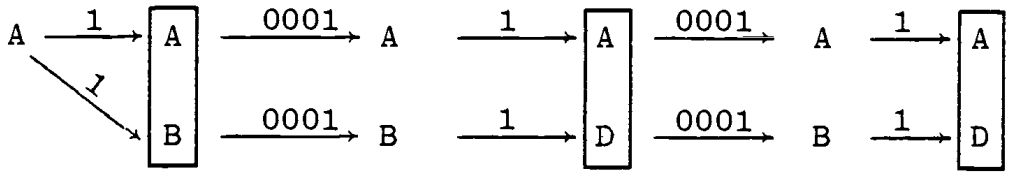


Figure 21. State transitions when 0001 is applied to the NFP (AB)

Corresponding to Figure 18, $I_1 = 1$, $S_i = A$, $S_j = A$, $S_{i1} = B$, in the above transition (Figure 21). Each of the state pairs enclosed in the rectangle corresponds to a NFP pair and the sequence (0001) corresponds to $q^{\ell-1}$. (A,10001) is the C-Test of length 5 and we will need only five tests to test the entire array for the faulty transitions $A \xrightarrow{1} B$ and $A \xrightarrow{1} D$. Assume another faulty transition $B \xrightarrow{0} B$, where the fault-free transition is $B \xrightarrow{0} C$. The transition in Figure 22 corresponds to that of Figure 19. The concurrent test (C-Test) is (B, 0001) and is of length 4. Hence only four tests are needed to test the entire array for this fault in any cell.

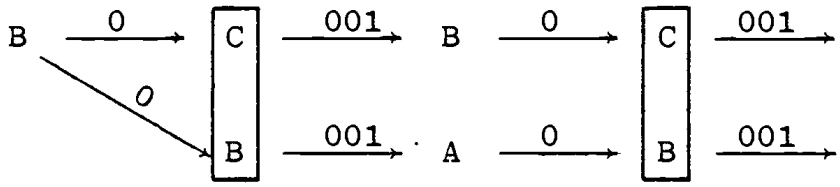


Figure 22. Application of 001 to the NFP (CB)

The sequence $q^{\ell-1}$ is such that starting from the pair (S_j, S_{i1}) the succeeding pairs of states (in response to $q^{\ell-1}$) are non-equivalent states. i.e., there is no convergence to the same state. Hence the following lemma.

LEMMA 3.2: A truncated C-Test where $\ell > p$ ('ℓ' is the length of the C-Test and 'p' is the number of cells in the array) will still detect the presence of the fault.

If the length of the C-Test is smaller than 'p' then the total number of tests needed to test the array depends only on the length of the C-Test sequence. However when $\ell > p$ the same test in truncated form will still detect the faulty transition (from lemma 3.2). But the number of tests needed to test the entire array will be dependent on p, the number of cells in the array.

3.6: Generation of C-Tests

At this juncture we will turn our attention to devising a procedure for the generation of C-Tests for all possible state faults in the flow table. We will be using the transition matrix extensively during the test-generation

procedure. Our objective is to derive a C-Test for a given state fault that behaves as shown earlier. The test generation procedure will be divided into two steps, one corresponding to $\ell \leq n$, and the other $\ell > n$, where ℓ is the length of the C-Test and n is the number of states in the flow table.

Definition: $w(e_{ij} \oplus e_{ik})$ corresponds to the weight of the exclusive-OR sum of the entries e_{ij} and e_{ik} and is defined as the number of 1's in the exclusive-OR sum. $w(\emptyset) = 0$, that is weight of a null entry is equal to 0.

Step 1: $\ell \leq n$

Let us first consider Figure 23. Here we have to find a sequence $q^{\ell-1}$ such that it transforms S_j into S_i and S_{i1} into S_{i2}, where $S_{i2} \neq S_i$. Since the length of the sequence $q^{\ell-1}$ cannot be determined a priori, we start with the transition matrix of power one (T) and check whether there is an entry (e_{ji}) that differs from the entry $e_{i1,i}$; i.e., whether there is an entry that satisfies the following weight relation (R1).

$$w(e_{ji} \oplus e_{i1,i}) \geq 1 \text{ ---------- (R1)}$$

If there is one then,

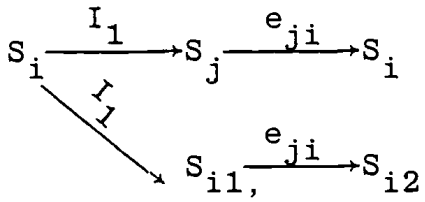


Figure 23. Application of e_{ji} to S_j and S_{i1}

At this stage we will define an $I_i(i=1,2,\ldots,m)$ equivalent state (I_i is the external vertical input) with respect to the state S_i. If k states on the application of external input I_i goes into the same state as S_i would on I_i, then all the k states are called I_i - equivalent states with respect to S_i and are donted as $I_iE(S_i)$.

eg.,

I / x	0	1
A	B	C
B	B	D
C	A	A
D	D	C

In the above flow table both A and B goes to B on an input 0 and hence we define A as a 0-equivalent state with respect to state B. This is denoted as OE(B) and similarly B is OE(A), A is 1E(D) and D is 1E(A).

Continuing our analysis we see from Figure 23, if S_{i2} happens to be $I_1\ E(S_i)$ then $I_1\ q^{\ell-1}$ cannot be used as a C-Test. On the other hand if S_{i2} is not a $I_1\ E(S_i)$ we will have the following transition where $q^{\ell-1}=e_{ij}$

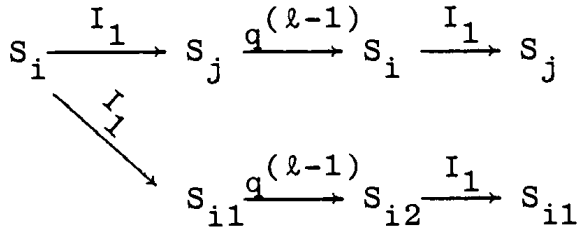


Figure 24. State transitions from (S_j S_i,) when $S_{i2} \neq I_1\ E(S_i)$

Because of the transition behavior in Figure 24, $I_1 q^{\ell-1}$ is a C-Test of length 2 and two tests will be sufficient to test the entire array. We now introduce an example that will serve to illustrate the generation of concurrent tests. The same flow table will be used throughout this section.

Example:

Flow Table: (F1)

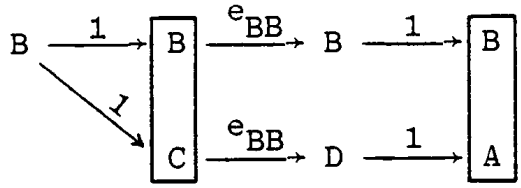
x \ I	0	1
A	B	A
B	C	B
C	D	D
D	C	A

(n=4)

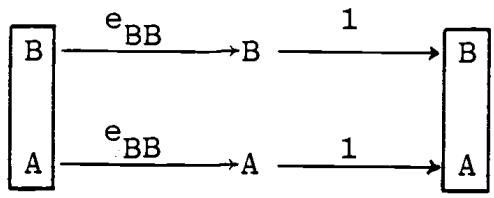
The T matrix corresponding to the above flow table is as follows

$$T = \begin{array}{c|cccc} & A & B & C & D \\ A & 1 & 0 & \emptyset & \emptyset \\ B & \emptyset & 1 & 0 & \emptyset \\ C & \emptyset & \emptyset & \emptyset & (0,1) \\ D & 1 & \emptyset & 0 & \emptyset \end{array}$$

B is equivalent to D on a 0 transition, hence is 0E(D), and similarly D is 0E(B), A is 1E(D) and finally D is 1E(A). Assume the faulty transition to be $B \xrightarrow{1} C$ and the corresponding fault-free transition is $B \xrightarrow{1} B$. In order to detect this fault, a C-Test $(1\ q^{\ell-1}) = 1e_{BB}(\ell \geq 1)$ should satisfy, first, the weight relation R1 and later the transition property of Figure 24. In the above T-Matrix we see that $w(e_{BB} \oplus e_{CB}) = 1$. i.e., $e_{BB} = 1$ and $e_{CB} = \emptyset$. Hence the following transition:



Since D is not a 1E(B) we proceed to find whether $e_{BB} = 1$ would satisfy relation (R1) with respect to e_{AB}. For $e_{BB}=1$ and $e_{AB}=0$ we have $w(e_{BB} \oplus e_{AB}) = 1$. Hence we have the following transition.



Therefore the C-Test is (B, 1 e_{BB}) =(B, 1 q) = (B, 1 1).

The length of this test is 2 and it detects the faulty transitions $B \xrightarrow{1} C$, $B \xrightarrow{1} A$. The total number of tests needed to test an entire array for these two faults is just two.

However we may not be able to generate all the C-Tests corresponding to all state faults using the T-Matrix of power one. Some of the faults may have test sequences of length $\ell > 2$. We then generate higher powers of the T-Matrix (up to the $(n-1)^{th}$ power), and proceed to devise tests as explained earlier. C-Test sequences generated up to this point have a maximum length $\ell = n$. Another example is presented here that involves the use of higher powers of T-Matrix.

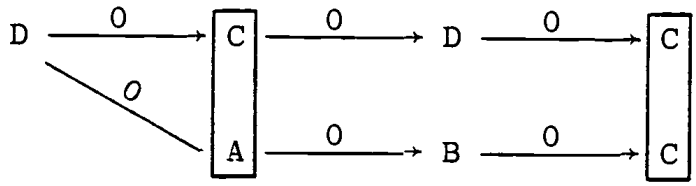
Example:

Consider a faulty transition $D \xrightarrow{0} A$, as opposed to the fault-free transition $D \xrightarrow{0} C$ in the flow table (F1). From the T-Matrix we find an e_{CD} that satisfies the relation R1.

i.e., for $e_{CD} = 0,1$ and $e_{AD} = \emptyset$

$$w(e_{CD} \oplus e_{AD}) = 1$$

Hence, for $e_{CD} = 0$,



The above transition shows that $(D, 0\ e_{CD}) = (D, 0\ 0)$ cannot be a C-Test because of the convergence. It can be shown similarly, that, using $e_{CD} = 1$ will also result in convergence. Hence we have to find the T^2 matrix to determine whether there is a C-Test for the given fault with $\ell = 3$.

$$T^2 = \begin{vmatrix} 11 & 10,01 & 00 & \emptyset \\ \emptyset & 11 & 10 & 00,01 \\ 01,11 & \emptyset & 00,10 & \emptyset \\ 11 & 10 & \emptyset & 00,01 \end{vmatrix}$$

[Note: We will henceforth drop the superscript k corresponding to the entries $(e_{ij})^k$ in T^k for sake of simplicity]

We find both e_{CD} and e_{AD} are $\emptyset$. i.e., no sequence of length two that will transform the states A and C to D. Proceeding further we determine T^3.

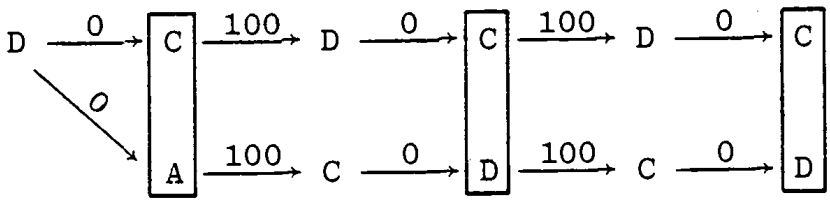
$$T^3 = \begin{vmatrix} 7 & 3,5,6 & 2,4 & (1,0) \\ 1,3 & 7 & 0,2,6 & 4,5 \\ 3,7 & 2,6 & \emptyset & (0,1,4,5) \\ 1,3,7 & 5,6 & 0,2,4 & \emptyset \end{vmatrix}$$

$(1,0) \rightarrow e_{AD}$

$(0,1,4,5) \rightarrow e_{CD}$

[Note: Entries in the T^3 matrix are decimal equivalents of the corresponding binary sequences of length 3]

We note that e_{CD}'s that satisfy the condition $w(e_{CD} \oplus e_{AD}) \geq 1$ in T^3 are 4(100) and 5(101). We then check whether (D, 0 e_{CD}) for e_{CD} = 4 can be a C-Test. The following transition shows that (D, 0100) is indeed a C-Test of length four and will detect the faults $D \xrightarrow{0} A$ and $D \xrightarrow{0} D$.



After generating the C-Test we determine all the state pairs between the NFP's corresponding to the sequence $q^{\ell-1}$. Since these pairs of states are responses to the sequence that is cyclic in nature, each such pair will appear at a periodicity of 'ℓ'. If $(q^{\ell-1}) = (a_1, a_2, \ldots, a_{\ell-1})$, $a_i \varepsilon (0,1)$ then we see from the following transition (Figure 25) that each of the pairs (S_{i3}, S_{i2}), (S_{i4}, S_{i5}),...are non-equivalent states and hence can be

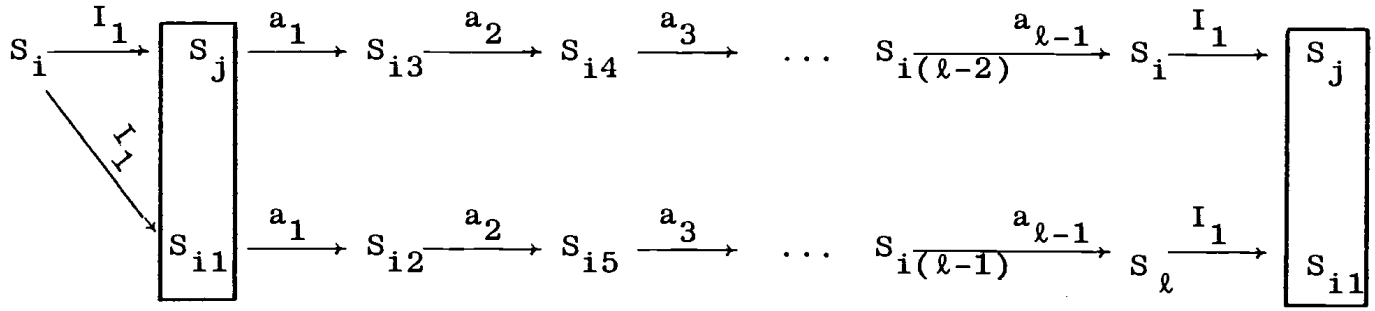
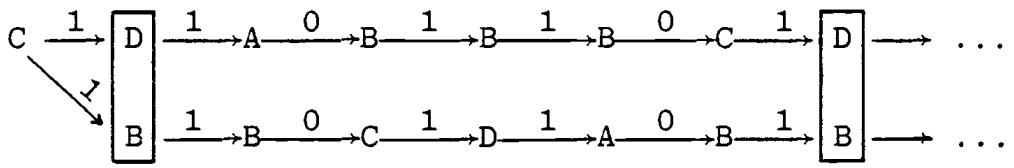


Figure 25. Application of $q^{\ell-1} = (a_1, a_2, \ldots a_{\ell-1})$ to the NFP $(S_j S_{i1})$

construed as NFPs. In essence the C-Test besides testing the faulty transition $S_i \xrightarrow{I_1} S_{i1}$, also tests for the faulty transformations $S_j \xrightarrow{a_1} S_{i2}$, $S_{i3} \xrightarrow{a_2} S_{i5}, \ldots$ $S_{i(\ell-2)} \xrightarrow{a_{\ell-1}} S_{i1}$ where the fault-free responses to S_j, $S_{i3}, \ldots, S_{i(\ell-2)}$ are S_{i3}, $S_{i4}, \ldots, S_i$ respectively. As an example, consider the flow table F1 and let $C \xrightarrow{1} B$ be the faulty transition, where $C \xrightarrow{1} D$ is the fault-free transition. The C-Test corresponding to the fault is (C, 110110) where $I_i = 1$ and $q^{\ell-1} = (10110)$.



From the transition diagram we see that the C-Test (C,110110) will also detect the faulty transitions $D \xrightarrow{1} B$, $A \xrightarrow{0} C$, $B \xrightarrow{1} D$, $B \xrightarrow{1} A$, $B \xrightarrow{0} B$ as opposed to the fault-free transitions $D \xrightarrow{1} A$, $A \xrightarrow{0} B$, $B \xrightarrow{1} B$, $B \xrightarrow{0} C$ in the (i+1),(i+2),(i+3), and $(i+4)^{th}$ cells in the one-dimensional array.

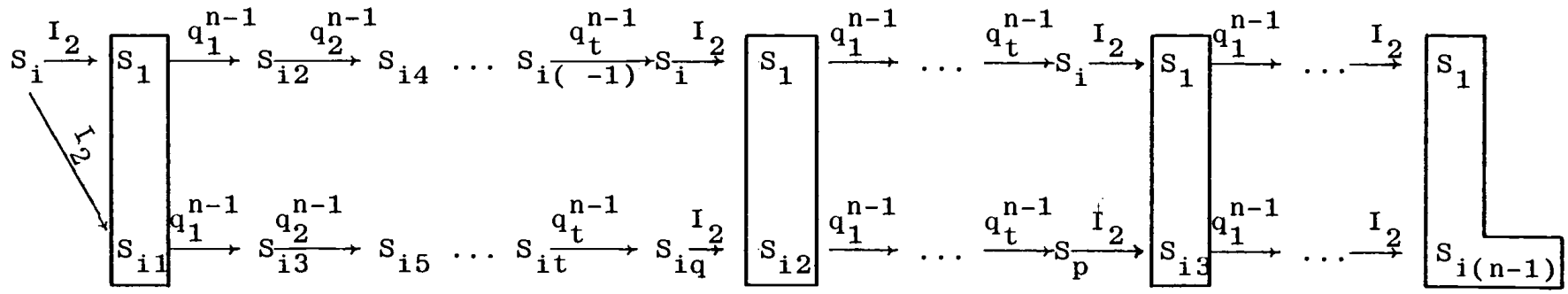
As said earlier, there may still be some faults for which test sequences will have a length greater than 'n'. In order to generate C-Tests for these faults we proceed to Step 2.

Step 2: $\ell > n$

In this step we address ourselves to the problem of finding a set of C-Tests for the remaining faults ($\ell > n$). Figure 26, an extension of Figure 18, represents the general case for ($\ell > n$) and will help us understand the nature of test design in this case.

Referring to Figure 26, let $S_i \xrightarrow{I_2} S_1$ be a fault-free transition and $S_i \xrightarrow{I_2} S_{i1}$ be the faulty transition. (q_1^{n-1}) is a sequence of length (n-1) enabling the transition from S_1 to S_{i2}. The subscript 't' corresponding to q^{n-1} denotes the order in which these q^{n-1} are determined. In practical cases it may not be necessary to go through so many state perambulations as in Figure 26. An upper bound on the subscript 't' will be later established as equal to 'n'.

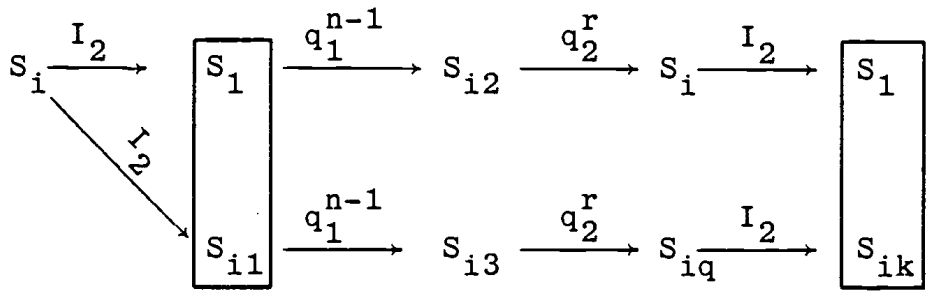
To facilitate the derivation of tests, Figure 26 is broken down as indicated in Figure 27. With reference to Figure 27, we first determine q_1^{n-1} from the T^{n-1} matrix such that $S_{i2} \neq S_{i3}$, where S_{i2} is arbitrary. We then use a T^r matrix ($r \leq (n-2)$) to determine a 'q_2^r' satisfying the transition requirement in Figure 27. The q's thus obtained obey the weight relation R1 given in Step 1.



q_i^{n-1} = sequence of length (n-1)

$t \leq n$

Figure 26. State transitions from (S_1 S_{i1}) when $\ell > n$.



$r = 1,2,\ldots,(n-2)$

$S_i \neq S_{iq}$, $S_1 \neq S_{ik}$

Figure 27. Application of $(q_1^{n-1} q_2^r I_2)$ to the NFP $(S_1 S_{i1})$.

If $S_{ik} = S_{i1}$ in the above Figure 27, we don't need to proceed further, and $(S_i, I_2\ q_1^{n-1}\ q_2^r)$ of length $(1+(n-1)+r)$ will be the C-Test corresponding to the faulty transition $S_i \xrightarrow{I_2} S_{i1}$. We now illustrate this using the following example.

Example:

Assume $D \xrightarrow{1} C$ and $D \xrightarrow{1} A$ to be the faulty and fault-free transitions in the flow table F1. Also assume no C-Test of length $(\ell \leq 4)$ exists for the above fault. In our case $n = 4$. Let us use T^3 matrix to find q_1^3 as in Figure 27.

$T^3 =$

	A	B	C	D
A	7	3,5,6	2,4	1,0
B	1,3	7	0,2,6	4,5
C	3,7	2,6	∅	0,1,4,5
C	1,3,7	5,6	0,2,4	∅

e_{AD}

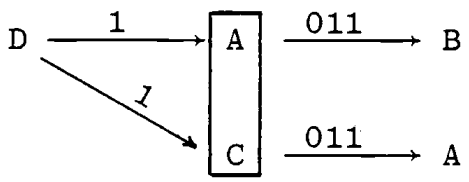
e_{CD}

In the T^3 matrix there is no e_{AD} that satisfies the condition

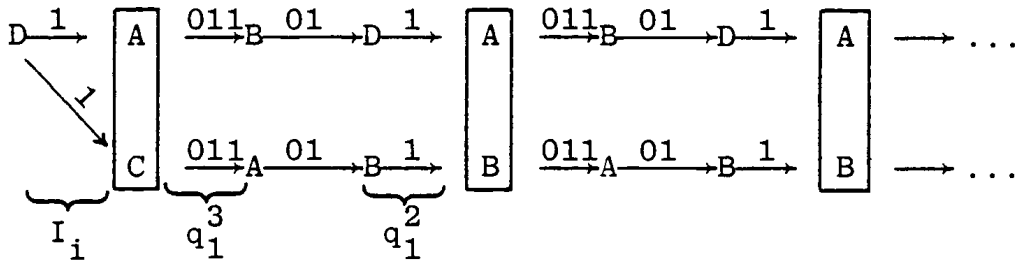
$$w(e_{AD} \oplus e_{CD}) \geq 1$$

Hence A is driven to some state other than D and from there back to D so that the above weight condition is satisfied. As we scan rows 1 and 3 (corresponding to states A and C) in the T^3 matrix we find that A cannot be driven to A and then to D since $e_{AA} = e_{CA} = 7$. Checking the next column we see that $e_{AB} = 3$ satisfies the weight condition $w(e_{AB} \oplus e_{CB}) \geq 1$. Hence $q_1^3 = e_{AB} = 3 = 011$.

i.e.,



Now we have to drive B to D and A to some state other than D using the same q_2^k ($k \leq (n-2)$). From T^2 we find an e_{BD} that satisfies $w(e_{BD} \oplus e_{AD}) \geq 1$, to be $e_{BD}=01=q_2^2$. (The T^2 matrix is given in page 72.) Hence we have the following transition.



The C-Test $(S_i, I_i\ q_1^3\ q_2^2)$ = (D, 101101) will detect the faults $D \xrightarrow{1} C$ and $D \xrightarrow{1} B$. Only seven tests will be required to test the one-dimensional array.

In Figure 27 if $S_{ik} \neq S_{i1}$ we apply the same sequence $(q_1^{n-1}\ q_2^r\ I_2)$ to the NFP (S_1S_{ik}) and note the response. If the response is the same as either one of the earlier two NFPs, i.e., (S_1S_{i1}) or (S_1S_{ik}), then we are done with the procedure. Otherwise we continue applying the sequence $(q_1^{n-1}q_2^rI_2)$ until one of the NFPs repeat. Whenever there is a convergence to the same state in the above procedure, $(I_2\ q_1^{n-1}\ q_2^r)$ cannot be used as a C-Test. At this stage we go back to Figure 26 and determine q_2^{n-1}, a sequence of length (n-1) from the T^{n-1} matrix such that $S_{i4} \neq S_{i5}$. We then find a sequence q_3^r from the T^r matrix ($r \leq (n-2)$) as before and check whether $(I_2q_1^{n-1}q_2^{n-1}q_3^r)$ can be a C-Test satisfying the aforesaid conditions. Similarly we proceed in the sense of Figure 26 until a test satisfying the C-Test

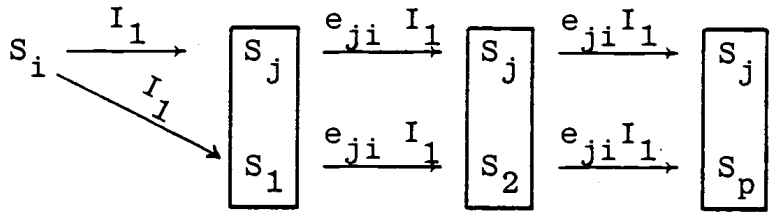
conditions is obtained.

This procedure as described will enable us to generate all the C-Tests whenever they exist. However such tests may not exist for all state faults. The following theorem states a necessary and sufficient condition for the existence of C-Tests.

THEOREM 3.2: Let $\{C_j^k\}$, $(k=1,2,\ldots,r)$, $r \geq 1$, be a set of closed walks with respect to the state S_j in the stage graph. The length of any C_j^k is greater than or equal to one. A C-Test exists for every NFP (S_jS_m), $(\forall m,\ m \neq j)$, iff

1. there is a $c_j^k \in \{C_j^k\}$ corresponding to every NFP such that the closed walk from S_j and a path from S_m (with the same label corresponding to c_j^k) do not intersect at the same vertex, and those c_j^k's that satisfy the above property should contain all the edges entering the vertex S_j and
2. whenever a new NFP (S_jS_t), $(t \neq m,j)$ is generated from (S_jS_m) due to c_j^k, such a c_j^k should satisfy condition 1 with respect to (S_jS_m).

Proof: Assume there is a C-Test for every NFP corresponding to S_j. Then we have the following transition



If $p = 1$ in the above transition, $c_j^k = e_{ji}I_1 \cdot e_{ji}I_1$ will test the NFPs (S_jS_1) and (S_jS_2). Since in each case the NFPs are non equivalent states the closed walk c_j^k from S_j and the paths from S_1 and S_2 have not intersected at the same vertex. In addition c_j^k contains the edge entering S_j, i.e., $c_j^k = e_{ji}I_1$ where the edge with the label I_1 enters S_j. From the assumption, there will be a C-Test for each NFP, and in each case we can prove, as above, the non intersection of edges at the same vertex.

The sufficiency proof is obvious and hence omitted.

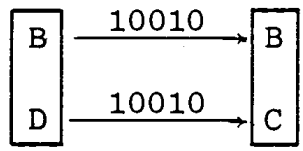
Q.E.D.

Although theorem 3.2 establishes the condition for the existence of a set of C-Tests capable of detecting all the faults, it should be noted that the existence of the graph-theoretic condition cannot be established a priori. If the condition of theorem 3.2 is not met, then there will not be a C-Test corresponding to that NFP. This is shown in the following example.

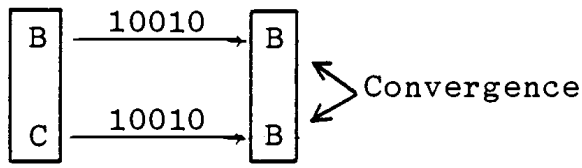
Example:

Let the faulty and fault-free transitions be $A \xrightarrow{0} D$ and $A \xrightarrow{0} B$, in the flow table F1. The corresponding NFP is (B D). A c_B^k starting with 0 cannot be a C-Test because $B \xrightarrow{0} C$ and $D \xrightarrow{0} C$. Similarly a c_B^k starting with 11 cannot be a C-Test. However there is a c_B^k (found from the transition matrix) that satisfies condition (1) of Theorem 3.2, and this corresponds to 10010.

i.e.,



A new NFP (B C) is generated due to 10010. Checking to see whether 10010 will satisfy condition (1) of Theorem 3.2 with respect to (B D) we find



intersection at the same vertex (B) and hence 10010 cannot be a C-Test. For the given fault we have found that there is no other c_B^k that will satisfy the conditions in Theorem 3.2. Hence no C-Test exists for the above fault.

While only certain flow tables have a complete set of C-Tests there are others which do not possess such tests for all faults. Although non-existence of tests cannot be

determined a priori, the corresponding flow tables can be modified in order that the flow table has tests for all faults. We now point out how such modification might be successfully accomplished.

As we saw earlier, non-existence of C-Tests is due to the intersection of a closed walk and a path at a single vertex. In terms of state transition, this would mean the convergence of the paths from the fault-free and faulty state into a single state corresponding to any input sequence. Such a flow table can be modified by adding an input column to the flow table so that no two states are the same in the new column, i.e., by adding a permutation column. This is illustrated in the following example.

Example:

Referring to the flow table of example (2) we see that the fault $A \xrightarrow{0} D$ doesn't have a C-Test. Let us introduce the modification and generate a C-Test for the above fault.

Original Flow Table:

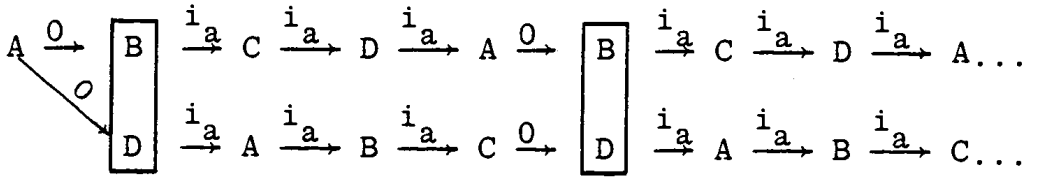
x \ I	0	1
A	B	D
B	C	A
C	D	A
D	C	B

Modified Flow Table:

x \ I	0	1	i_a
A	B	D	B
B	C	A	C
C	D	A	D
D	C	B	A

←ADDED COLUMN

Using the modified flow table we can arrive at the following transition.



And hence $(A, 0i_a i_a i_a)$ will detect the fault $A \xrightarrow{0} D$.

3.7: Bounds on the Length and Total Number of Tests

Let $S_i \xrightarrow{I_d} S_j$ be a fault-free transition and $S_i \xrightarrow{I_d} S_k$ be a faulty transition. If $(I_d\, q^{\ell-1})$ is a C-Test then $S_j \xrightarrow{q^{\ell-1}} S_i$ and $S_k \xrightarrow{q^{\ell-1}} S_p$ where S_p is not a $I_d E(S_i)$.

Since the given flow table is reduced and strongly connected there is an input 'y_1' that distinguishes S_j and S_k. If the response to y_1 is not $(S_i S_p)$ we continue applying inputs $y_2, y_3, \ldots, y_r$ until we have a $(S_i S_p)$ as the response to $(S_j S_k)$. We know that there are only $n(n-1)$ state pairs corresponding to an n-state flow table. Hence the length of such a sequence of y_i inputs need not exceed $(n(n-1)-1)$. So $q^{\ell-1} \leq (n(n-1)-1)$ and the length of the C-Test $\ell \leq 1 + (n(n-1)-1) = n(n-1)$.

For an n-state flow table we have $n(n-1)$ faulty state transitions. If there is a C-Test for each fault then the upper bound on the number of tests needed to test for a fault in the one dimensional array is $\ell = (n(n-1))$.

Hence the total number of tests - assuming $\ell<p$ for all C-Tests - needed to test the array for all state faults will not exceed (mn(n-1) n (n-1)). The complexity on the upper bound is $O(mn^4)$ which is not dependent on 'p', the number of cells in the array. However if, say, 'r' state faults have C-Tests of length $\ell>p$, then the upper bound on the total number of tests will be $\{\{mn(n-1) - r\}\ \{n(n-1)\} + rp\}$ indicating the dependence on the number of cells in the array when $\ell>p$ for some C-Tests.

3.8: Algorithm for the Generation of C-Tests

The algorithm is divided into two parts, one for $\ell \leq n$ and the other for $\ell > n$. Following is an explanation of the notations in the flow chart for Part (1) of the algorithm.

Notations:

$\{G_{ij}^{I_d}\}$ --- List of NFPs corresponding to the fault-free transition $S_i \xrightarrow{I_d} S_j$. $(d=1,2,\ldots,m)$, $(i=1,\ldots,n)$.

$S_j S_k \{G_{i_j}^{I_d}\}$, $(k \neq j)$

$\{I_d\}$ --- External input 'y'

$\{T_{ij}^{I_d}\}$ --- C-Test set for the NFPs' in $\{G_{ij}^{I_d}\}$

$\{L_r\}$ --- A set of NFPs $(r=1,\ldots,Q)$ that, on application of q_r^k (a binary sequence

of length k, $k \leq (n-1)$) produces the same response $S_i S_p$, $i \neq p$. I.e., $\forall\, S_j S_k \in \{L_r\}$

$$S_j \xrightarrow{q_r^k} S_i \quad \text{and} \quad S_k \xrightarrow{q_r^k} S_p$$

$\{L_r'\}$ --- Set of NFPs that are responses to $S_i S_p$ on an input I_d i.e.,

$$\{L_r\} \xrightarrow{q_r^k} (S_i S_p) \xrightarrow{I_d} \{L_r'\}$$

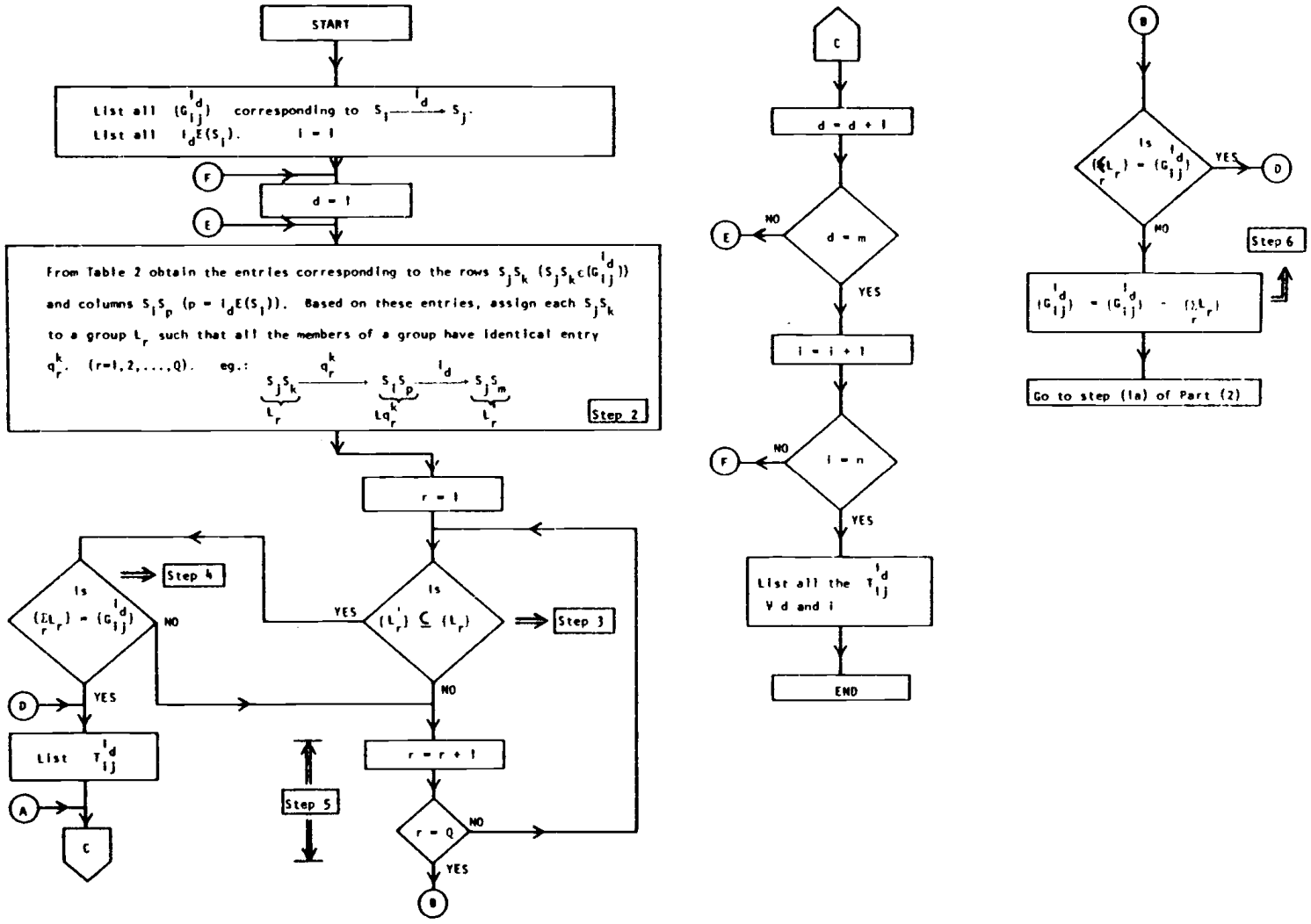
Table 1 --- Obtained from T^{n-1} matrix. It has 2^{n-1} rows and $\binom{n}{2}$ columns. Rows represent binary input sequences of length (n-1). Columns represent input state pairs $(S_i S_j)$. The entries in the table represent the mapping $S_i S_j \times Y \longrightarrow \hat{S}_i \hat{S}_j$ (output state pairs). Y corresponds to the binary input sequence in the rows.

Table 2 --- Obtained from $T, T^2, \ldots, T^{n-1}$ matrices. Has $\binom{n}{2}$ rows and n(n-1) columns. Rows represent input state pairs $(S_i S_j)$. Columns are output state pairs $(\hat{S}_i \hat{S}_j)$. Entries in the table are binary sequences of varying length 'k'

$(k=1,2,\ldots,(n-1))$ that satisfy the mapping $(S_iS_j) \times Y \longrightarrow (\hat{S}_i\hat{S}_j)$

(A final note on the entries in Tables 1 and 2 --- The blank entries in Table 1 correspond to convergence to the same states and is analogous to satisfying the weight relation R1. Similarly the mapping relation in Table 2 satisfies R1.)

FLOW CHART FOR PART (1) OF C-TEST GENERATION ALGORITHM



(FLOW CHART EXPLANATION):

Step 2: Instead of checking the weight relation R1 (Section 3.5) for every faulty transition, corresponding to a fault-free transition, we use Table (2) which contains all the entries from the T^k matrices (k=1,...,n-1). In other words, Table 2 contains entries that satisfy the weight relation R1. Find q_r^k, an entry from Table 2 such that $\forall\, S_jS_k \in \{L_r\}$ the S_jS_k have the same q_r^k.

$$\text{i.e., } \underbrace{\{S_jS_k\}}_{L_r} \xrightarrow{q_r^k} \underbrace{(S_iS_p)}_{L_r\ q_r^k} \xrightarrow{I_d} \underbrace{\{S_jS_m\}}_{L_r'}$$

Step 3: If $S_jS_m \in \{L_r\}$ then $S_jS_m \xrightarrow{q_r^k} S_iS_p$, and we will have $(S_i, I_d\ q_r^k)$ as the C-Test for all the NFPs in L_r. Hence we check the covering relation $\{L_r'\} \subseteq \{L_r\}$ in this step

Step 4: Check whether we have C-Tests for all the NFPs in $\{G_{ij}^{I_d}\}$

Step 5: If the covering relation in step 3 is not satisfied, or if we haven't generated tests for all the NFPs, the next q_r^k is checked for the covering

relation of step 3 until we have exhausted all q_r^k's

Step 6: If we don't have tests for all NFPs in $\{G_{ij}^{I_d}\}$, we remove from this list those NFPs which have a C-Test and proceed to Part (2) of the algorithm to generate C-Tests for the remaining NFPs. If each one of these has a C-Test, then the length of each will be greater than 'n'.

Algorithm: Part (2) (ℓ>n)

1a.) t = 1

1b.) Let there be R (R≤n-1) NFP pairs in $\{G_{ij}^{I_d}\}$.

For each of these NFP obtain $\{M_R^1\}$ from Table (1).

$(q_1^{n-1}, S_{rm}^1 S_{1m}^1) \varepsilon \{M_R^1\}$

i.e.,

$$\underbrace{S_j S_k}_{\{G_{ij}^{I_d}\}} \xrightarrow{q_1^{n-1}} S_{rm}^1 S_{1m}^1$$

$$\{G_{ij}^{I_d}\} \longleftarrow \{M_R^1\} \longrightarrow$$

2. This step is performed for (t>1)

$\{M_R^t\} = \{M_R^t\} - \{M_R^{t-1}\}$; for t = 1, $\{M_R^1\} = \{M_R^1\}$

The above set-subtraction is performed for the following reason: If there is no sequence of

length $\ell \leq n-1$ that would take the states $(S_{rm}^{t-1} S_{\ell m}^{t-1}) \varepsilon \{M_R^{t-1}\}$ to $S_i S_p$, then these state pairs can be deleted if they are present in $\{M_R^t\}$.

3.) Check whether $\{M_R^t\} = \emptyset$. If so go to (A) in Part (1) of the flow chart. If not, for each state pair in $\{M_R^t\}$ and $\forall$ R obtain the entries q_2^r from Table (2). Figure 28 illustrates this procedure, and the process is similar to step (2) in the flow chart for Part (1).

4.) $\forall$ r determine $S_i S_p \xrightarrow{I_d} L_r^1, L_r^2, \ldots, L_r^s$; $(s \leq n-1)$. Verify for $\forall$ r whether $L_r^s \subseteq L_r^p$, $p \leq s-1$. This is the same as step (3) in the flow chart of Part (1).

5.) NFPs corresponding to $\Sigma_p L_r^p$ for 'p' satisfying the covering condition in (4) will have a C-Test $(S_i, I_d q_1^{n-1} q_2^r)$, where q_1^{n-1} is of length (n-1) and q_2^r is of length less than (n-1).

6.) If $\{\Sigma_p L_r^p\} = \{G_{ij}^{I_d}\}$ then go to (A) in Part (1) of the flow chart. This means that we have found

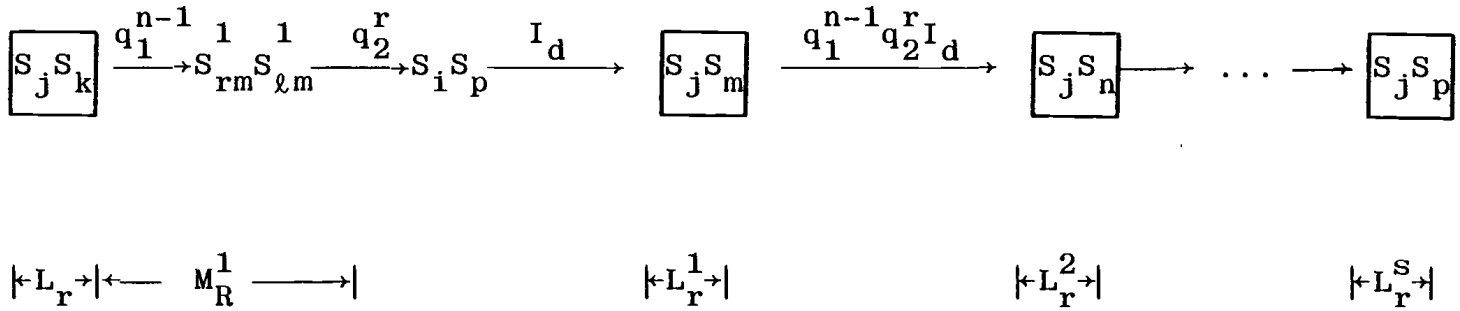


Figure 28. Application of $(q_1^{n-1} q_2^1 I_d)$ for the NFP $(S_j S_k)$.

tests for all the NFPs we started with in this part. If not, then set $t = t + 1$. Since the length of the test sequence is bounded by $n(n-1)$ and the length of each 'q' is not greater than $(n-1)$, 't' is also bounded. I.e., $t \leq \frac{n(n-1)}{(n-1)} = n$. Hence check if $t>n$. If so, go to (A) in Part (1) of the flow chart. Otherwise go to Step (2) and follow the transition given below:

$$S_j S_k \xrightarrow{q_1^{n-1}} S^1_{rm} S^1_{\ell m} \xrightarrow{q_2^{n-1}} S^2_{rm} S^2_{\ell m} \longrightarrow \ldots \xrightarrow{q_t^{n-1}} S^t_{rm} S^t_{\ell m}$$

$$\vdash\!\!\longrightarrow M_R^1 \longrightarrow\vdash\!\!\longrightarrow M_R^2 \longrightarrow \qquad\qquad \vdash\!\!\longrightarrow M_R^t \longrightarrow$$

3.9: Examples

The algorithm for Part (1) is illustrated in the section with a complete example.

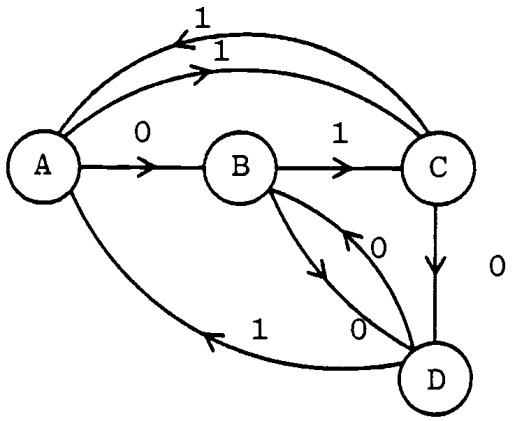
Example 1:

Design a one-dimensional, combinational, iterative array that will output a 1 whenever it recognizes the following regular expression, $((011)^* + (11)^*)((010 + 10 + 00)((010)^* + (1010)^* + (100)^*)^*)$. Generate C-Tests for all state faults whenever they exist.

Flow Table:

x \ y	0	1
A	B	C
B	D	C
C	D	A
D	B	A

Flow Graph:



Part (1):

Step (1): i = 4, d = 2 (2 input (y) values),

$I_1 = 0$, $I_2 = 1$

List of $G_{ij}^{I_d}$:

G_{AB}^{0}	G_{BD}^{0}	G_{CD}^{0}	G_{DB}^{0}
BA, BC, BD	DA, DB, DC	DA, DB, DC	BA, BC, BD
G_{AD}^{1}	G_{BC}^{1}	G_{CA}^{1}	G_{DA}^{1}
CA, CB, CD	CA, CB, CD	AB, AC, AD	AB, AC, AD

List of $I_dE(S_i)$:

OE(A)	OE(D)	OE(B)	OE(C)
D	A	C	B

1(EA)	1(EB)	1(E(C)	1E(D)
B	A	D	C

Generation of Transition Matrices for the given Flow Table:

$$T = \begin{vmatrix} \emptyset & 0 & 1 & \emptyset \\ \emptyset & \emptyset & 1 & 0 \\ 1 & \emptyset & \emptyset & 0 \\ 1 & 0 & \emptyset & \emptyset \end{vmatrix} \qquad T^2 = \begin{vmatrix} 3 & \emptyset & 1 & 0,2 \\ 1,3 & 0 & \emptyset & 2 \\ 1 & 0,2 & 3 & \emptyset \\ \emptyset & 2 & 1,3 & 0 \end{vmatrix}$$

$$T^3 = \begin{vmatrix} 1,3,5 & 0,4,6 & 7 & 2 \\ 5 & 2,4,6 & 1,3,7 & 0 \\ 7 & 2 & 1,3,5 & 0,4,6 \\ 1,3,7 & 0 & 5 & 2,4,6 \end{vmatrix}$$

Note: The entries in the matrices are decimal equivalents of binary sequences of length 'k', where 'k' is the power of the matrix.

TABLE (1): OBTAINED FROM T^3 MATRIX

INPUT SEQUENCES ↓ / INPUTE STATE PAIRS →	AB	AC	AD	BC	BD	CD
0	BD	BD	-	-	DB	DB
1	AC	AC	-	-	CA	CA
2	DB	DB	-	-	BD	BD
3	AC	AC	-	-	CA	CA
4	-	BD	BD	BD	BD	-
5	-	AC	AC	AC	AC	-
6	-	BD	BD	BD	BD	-
7	-	CA	CA	CA	CA	-

TABLE (2): OBTAINED FROM T, T^2, T^3 MATRICES

INPUT STATE PAIRS ↓ / OUTPUT STATE PAIRS →	AC	BD	CA	DB
AB	001,011	0,000	01	00,10 010
AC	11,001 011,101	0,000 100,110	1,01 111	00
AD	11,101	100,110	1,111	10
BC	11,101	100,110	1,111	10
BD	01,11 101	00,010 100,110	1,001 011,111	0,10 000
CD	01	00,010	001,011	0,000

Note: Since transitions from the state pairs in the rows to AB, AD, BA, BC, CB, CD, DA and DC do not exist, corresponding columns are omitted from the above table, Table (2). Entries with respect to transitions from BA, CA, CB, DA, DB, DC can be found as follows. If, for example, entries with respect to the transition from BA to (say) DB are needed, we look for entries in the row AB and column BD since AB$\longrightarrow$BD is same as BA$\longrightarrow$DB.

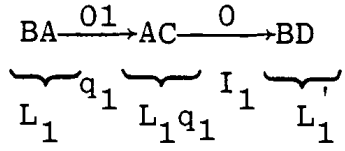
Step 2: Start with i = 1, d = 1, i.e., $I_d = 0$

$A \xrightarrow{0} B$ fault-free transition

$\{G^0_{BA}\} = (BA, BC, BD)$

Using the entries in Table (2), the groups L_r are formed as follows. The L_r correspond to rows in the table.

	L_r	q_r	$L_r q_r$	L_r'
r = 1	BA BD	01	AC	BD
r = 2	BC BD	11	AC	BD
r = 3	BA BD	101	AC	BD



Step (3):

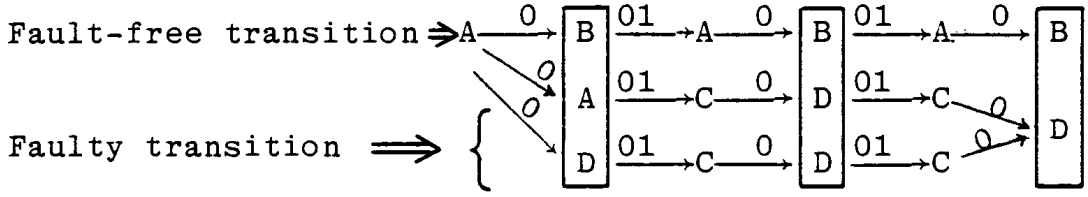
$L_1 = (BA, BD)$; $L_1' = (BD)$

$L_1' \subset L_1$

Hence the C-Test for the NFPs in

L_1 is $(A, I_1\, q_1)$ which is (A, 001)

Check:



Step (4):

Check whether $\sum_r L_r = \{G^0_{AB}\}$

$L_1 \neq G^0_{AB}$

Step (5):

Hence $r = r + 1 = 2$

$r \neq Q \neq 3$

Step (3):

$L_2 = (BC, BD)$; $L_2' = (BD)$

$L_2' \subset L_2$

Hence C-Test for L_2 is $(A, I_1 q_2) = (A,011)$

Step (4):

$\Sigma_r L_r = L_1 + L_2 = (BA, BC, BD) = \{G^0_{AB}\}$

C-Test set for $\{G^0_{AB}\} = \{T^0_{AB}\} = \{(A,001),(A,011)\}$

Step (2):

$d = 2$, $I_d = I_2 = 1$

$A \xrightarrow{1} C \Rightarrow$ fault-free transition

$\{G^1_{AC}\} = (CA, CB, CD)$

from Table (2):

	L_r	q_r	$L_r q_r$	L_r'
r = 1	CA CB	1	AC	CA
r = 2	CA CD	01	AC	CA
r = 3	CA CB	111	AC	CA

Step (3):

$L_1' \subset L_1$; $L_2' \subset L_2$; C-Test for L_1 is $(A, I_2 q_1) = (A, 11)$

C-Test for L_2 is $(A, I_2 q_2) = (A, 101)$

Step (4):

$L_r = L_1 + L_2 = \{G^1_{AC}\}$

$\{T^1_{AC}\} = \{(A,11),(A,101)\}$

Step (1):

$i = 2,\ d = 1,\ I_1 = 0$

$B \xrightarrow{0} D \Longrightarrow$ fault-free transition

$\{G^0_{BD}\} = (DA,\ DB,\ DC)$

Step (2):

From Table (2):

	L_r	q_r	$L_r q_r$	L_r'
r = 1	DB DC	0	BD	DB
r = 2	DA DB	10	BD	DB
r = 3	DB DC	000	BD	DB

Step (3):

$L_1' \subset L_1$; $L_2' \subset L_2$; C-Test for L_1 $(B,I_1q_1)=(B,00)$

C-Test for L_2 $(B,I_1q_2)=(B,010)$

Step (4):

$$L_r = L_1 + L_2 = \{G^0_{BD}\} \; ; \quad \{T^0_{BD}\} = \{(B,00),(B,010)\}$$

In a similar manner we can generate C-Tests for all the $\{G^{I_d}_{ij}\}$'s whenever they exist. For this example we have C-Tests for all the NFPs. Following is the list of C-Tests generated using Part (1) of the algorithm.

$T^0_{AB} = \{(A,001),(A,011)\}$ $\quad$ $T^0_{CD} = \{(C,011),(C,001)\}$

$T^1_{AC} = \{(A,11),\ (A,101)\}$ $\quad$ $T^1_{CD} = \{(C,11),(C,101)\}$

$T^0_{BD} = \{(B,00),(B,110)\}$ $\quad$ $G^0_{DB} = \{(D,00),(D,010)\}$

$T^1_{BC} = \{(B,110),(B,100)\}$ $\quad$ $T^1_{DA} = \{(D,100),(D,110)\}$

Since each test is cyclic in nature, the number of tests necessary for testing a one-dimensional array will be equal to the length of the test sequence. For the given example the number of tests needed to test all the cells in the one-dimensional array is 22, well below the upper bound stated earlier.

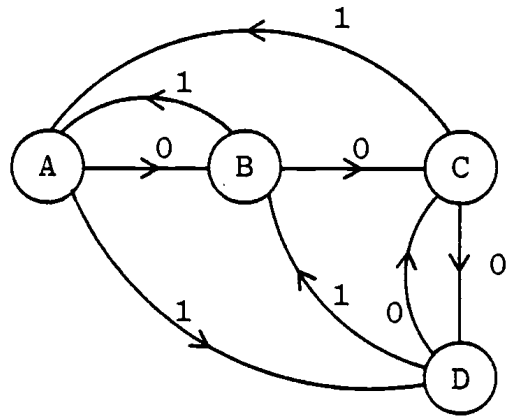
The algorithm for Part (2) is illustrated in this example.

Example (2):

Construct a one-dimensional, iterative array that recognizes the following regular expression, $((01)^* + (001)^*)((000 + 1)((111)^* + (100)^* + (011)^* + (00)^*)^*)$. Generate C-Tests for the array whenever they exist.

Flow Table:

x \ y	0	1
A	B	D
B	C	A
C	D	A
D	C	B



Part (1):

Step (1):

$$T = \begin{vmatrix} \emptyset & 0 & \emptyset & 1 \\ 1 & \emptyset & 0 & \emptyset \\ 1 & \emptyset & \emptyset & 0 \\ \emptyset & 1 & 0 & \emptyset \end{vmatrix}, \quad T^2 = \begin{vmatrix} 1 & 3 & 0,2 & \emptyset \\ 1 & 2 & \emptyset & 0,3 \\ \emptyset & 2,1 & 0 & 3 \\ 1,2 & \emptyset & 2 & 0 \end{vmatrix}$$

$$T^3 = \begin{vmatrix} 1,5,7 & 2 & 6 & 0,3,4 \\ 5 & 1,2,7 & 0,4,6 & 3 \\ 1,3,5 & 7 & 2,4,6 & 0 \\ 5 & 1,2,6 & 0 & 3,4,7 \end{vmatrix}$$

Table (1): Obtained from T^3 matrix

	AB	AC	AD	BC	BD	CD
0	DC	-	DC	CD	-	DC
1	AB	-	AB	BA	-	AB
2	-	BC	-	BC	-	CB
3	-	DA	-	DA	-	AD
4	DC	DC	-	-	CD	CD
5	-	-	-	-	-	-
6	-	-	CB	-	CB	CB
7	AB	AB	AD	-	BD	BD

Table (2):

	AB	AD	BA	BC	BD	CB	CD	DA	DB	DC
AB	001 111			0	11	10	00	1		000 100
AC	01 111			010	0,11	10		1 011		100
AD	001	111	11	0		110	00		1	000
BC	01		001	010			0 000			00 011
BD	1			10	111	110	100	11		
CD	1 001	011	01	10	111	010 110	100 00	11		0 000

Step (2):

$i = 1,\ d = 1,\ I_1 = 0$

$A \xrightarrow{0} B$

$\{G^0_{AB}\} = (BA,\ BC,\ BD)$

From Table (2):

	L_r	q_r	$L_r q_r$	L_r'
r = 1	BA	1	AD	BC
	BD		AB	
r = 2	BC	01	AB	BC

Step (3):

$L_1' \not\subset L_1$. Hence $r = r + 1 = 2$.

$L_2' \subset L_2$

Hence the test for L_2 is $(A, I_1 q_2) = (A, 001)$

Step (4):

Check whether $\Sigma_r L_r = \{G^0_{AB}\}$

$$L_2 \neq (BC) = \{G^0_{AB}\}$$

Step (6):

$$\{G^0_{AB}\} = \{G^0_{AB}\} - \{L_2\} = (BA, BD)$$

Go to step (1a) of Part (2)

Step (1b): Part (2)

R = 2 since there are two NFP pairs (BA, BD) that do not have C-Tests at this stage

From Table (1):

M_1^1 :

q_1^3	BA
0	CD
1	BA
4	CD
7	BA

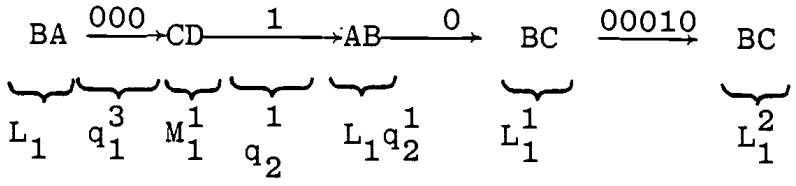
M_2^1 :

q_2^3	BD
4	CD
6	CB
7	BD

Step (3):

From Table (2):

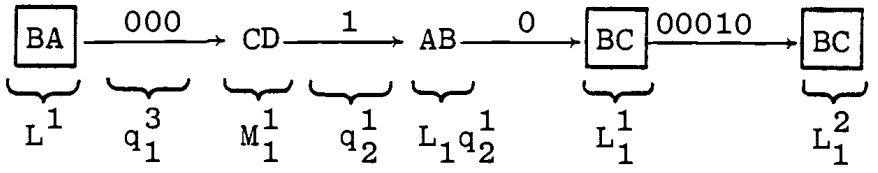
	L_r	q_1^3	M_1^1	M_2^1	q_2^r	$L_r q_2^r$	L_r^1	L_r^2
r = 1	BA	000	CD	-	1	AB	BC	BC
	BA	100	CD	-				BB
	BD	100	-	CD				BB
r = 2	BA	000	CD	-	001	AB	BC	BC
	BD	110	-	CB				BB



Step (4):

In the block L_1 we have one transition that satisfies the covering relation $L_1^2 \subset L_1'$.

i.e.,



Step (5):

Corresponding NFP is (BA), and the C-Test is $(A, I_1\ q_1^3\ q_2^1) = (A, 00001)$. Incidentally this also tests for the NFP (BC).

Step (6):

With respect to the NFP(BD), L_2^2, L_1^2 are (BB) indicating convergence, and no test exists at this stage for the NFP (BD). t = t + 1 = 2

Go to step (2)

Step (2):

M_2^1 :

q_1^3	BA
4	CD
6	CB
7	BD

M_2^2 : Obtained from Table (1)

q_2^3	CD
0	DC
1	AB
2	CB
3	AD
4	CD
6	CB
7	BD

q_2^3	CB
0	DC
1	AB
2	CB
3	AD

q_2^3	BD
4	CD
6	CB
7	AD

Delete from $\{M_2^2\}$ all the elements contained in $\{M_2^1\}$ since we found earlier that BD cannot be driven through the states in $\{M_2^1\}$ in order to generate a C-Test.

Hence $\{M_2^2\} = \{M_2^2\} - \{M_2^1\}$

$\{M_2^2\}$:

q_2^3	CD
0	DC
1	AB
3	AD

q_2^3	CB
0	DC
1	AB
3	AD

Step (3):

r=1: BD $\xrightarrow{100}$ CD $\xrightarrow{000}$ DC $\xrightarrow{01}$ AB $\xrightarrow{0}$ BC $\xrightarrow{100000010}$ BB

L_1 q_1^3 q_2^3 q_3^1 L_1^1 L_1^2

We do not have a test at this point because of the convergence seen above (L_1^2 = BB). For all values of 'r' it has been found that there is convergence. Proceeding further we find that all the members of $\{M_2^3\}$ are contained in $\{M_2^2\}$. After performing step (2) we have $\{M_2^3\} = \emptyset$. We return to (A) in Part (1).

However C-Tests do exist for all other NFPs and they are (generated using Part (1) and (2) of the algorithm) as follows:

C-Test Set: (A,001);(A,111);(B,001);(B,111);(C,00); (C,010)(C,100);(C,11000);(D,00);(D,00100);(D,111);(D,100). In order to test the NFP (BD) we will need 'p' tests where 'p' is the number of cells in the array. The total number of tests needed is thus (38 + p).

3.10: Fault Detection in Specialized Arrays

Until now we have discussed test generation procedures for iterative arrays (cellular) where all the cells in a given array realize the same function. However, there are also one-dimensional arrays called Maitra cascades [23] and Cut-Point Cellular arrays [30], where each cell realizes a different function. Essentially, the cells in both cases realize a two variable function. In case of a Maitra cascade, each cell is flexible, in that the logical elements in the cell can be set up to produce any one of the sixteen functions of its inputs. Maitra has shown that, even though each cell is a general-function cell, it is not possible to produce an arbitrary function of n variables at the output of the end cell n if n>2. Figure 29 is an example of a simple Maitra cascade where each cell has two inputs and a single input.

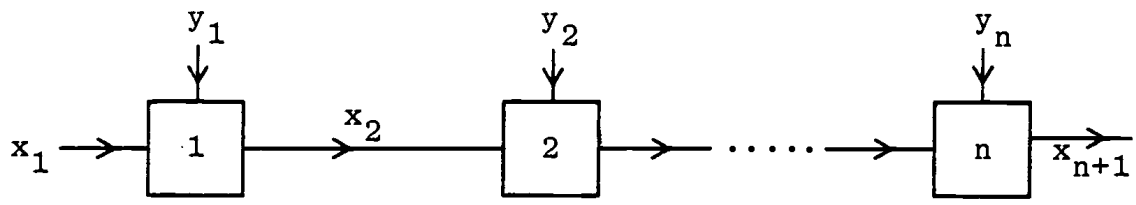


Figure 29. Maitra Cascade

In order to test the Maitra cascade for multiple faults, the algorithm for one-dimensional test generation cannot be applied directly. This is due to the fact that

each cell in the Maitra cascade realizes a different function. Instead we will adopt the following strategy.

1. Generate a T matrix for each cell individually and let's call them $T_1, T_2, \ldots, T_n$.
2. Perform the following matrix multiplication.

$$T_{n+1} = T_1 \times T_2 \times \ldots \times T_n$$

(T_{n+1} is a transition matrix where each entry $(y_1 y_2 \ldots y_n)$ is of length n. Since there are only two states corresponding to any cell realization, an entry say, e_{11} in the T_{n+1} matrix represents a sequence of y inputs that will transform S_1 applied at the first cell into S_1 at the output of the last cell (n^{th} cell)).

3. Any multiple-state fault inside the cascade will produce either a faulty or normal output at the output of the end cell (n). In the latter case the faults are said to be masked and cannot be detected (Theorem 3.1). This permits us to omit the testing of every individual cell (every T_i) in the cascade. Hence it will be sufficient to test for all state faults (as before) in the T_{n+1} matrix, thus guaranteeing multiple-fault-detection in the Maitra cascade.

Since all switching functions cannot be produced by a Maitra cascade, Minnick [30] proposed a generalization of the Maitra cascade that was functionally complete -

that is, can generate all n-variable functions. His realization, called the cut-point cellular array, was a two-dimensional arrangement of cells, each cell realizing a restricted set of two variable function ($\bar{y}$, $\bar{x}+\bar{y}$, $\bar{x}\bar{y}$, $x+y$ $x\bar{y}$ $x \oplus y$) as shown in Figure 30. The horizontal input to each cell is taken from a bus bar. The required function $f(x_1,x_2,\ldots,x_n)$ is realized as $f_1(x_1,x_2,\ldots,x_n)$ + $f_2(x_1,x_2,\ldots,x_n)$ + ... + $f_m(x_1,x_2,\ldots,x_n)$, where each f_i is realized by the corresponding i^{th} column. In order to get the arbitrary f, each of the terms f_i is ORed, using m rotated cells, each of which realizes the OR function.

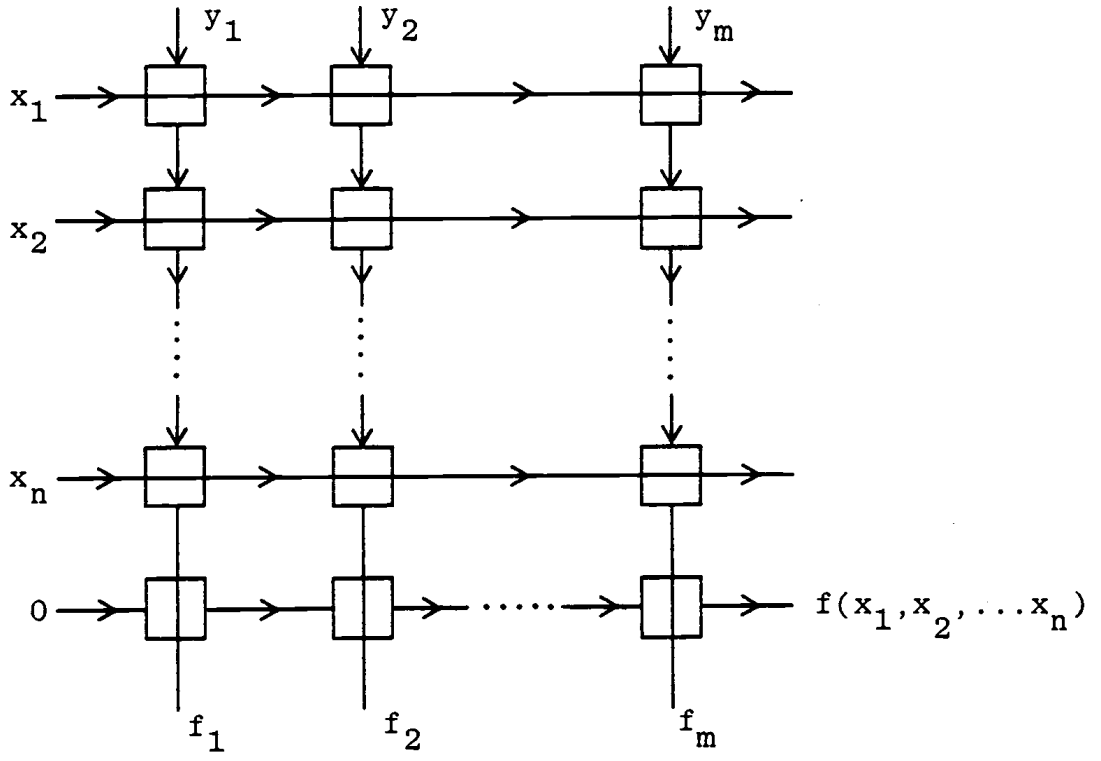


Figure 30. Cut-point cellular array.

Testing cut-point cellular arrays for multiple faults is similar to that of testing the Maitra cascade. Each column in the cut-point array is similar to a Maitra cascade and is tested as described earlier. If the columns are fault-free, we then have to test the collector row (bottom row). Consider the cut-point array in Figure 31 that is capable of generating all three variable functions. An arbitrary three variable function can be represented as follows.

$$f(x_1,x_2,x_3) = x_3\overline{G}(x_1, x_2) + \overline{x}_3\overline{H}\ (x_1,x_2)$$

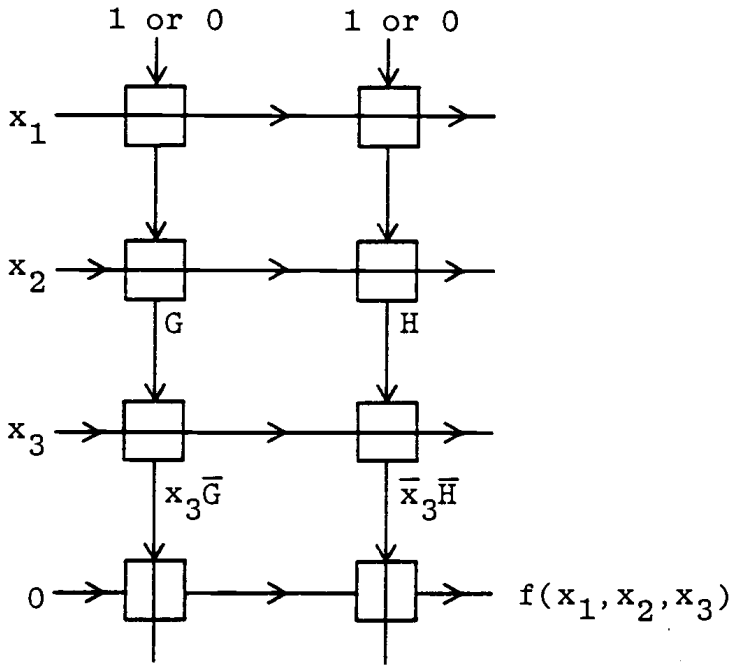


Figure 31. Cut-point cellular array realizing $f(x_1,x_2,x_3)$.

The output of each cell in the third row in Figure 31 is a product term of three variables. If there is an input

combination $(x_1,x_2,\ldots,x_n)\varepsilon\{0,1\}^n$ for which all $f_i(x_1,x_2,\ldots,x_n) = 0.$, then the input $(x_1,x_2,\ldots,x_n)$ can be applied to the array in order to test the bottom row, that is, we will make all vertical inputs to the bottom row equal to 0 and if we set its left most input to 0, the entire row can be tested for all multiple faults, [35].

3.11: Conclusion

In the preceeding section we have developed an algorithm for the generation of tests that will detect multiple-faults in a one-dimensional iterative array. The nature of the test is such that generally more than one cell will be tested concurrently. This results in the reduction in the total number of necessary tests. However it has been shown by means of a counterexample that such tests do not exist for all flow tables. In our analysis of the nature of faults we have been able to define a larger class of faults, (i.e. larger than the "stuck-at" fault class), namely state faults that cover all single and multiple faults within the cell. Tests derived under this assumption also detect faults in the inter-cell leads.

It should be noted that no mention of external outputs is made throughout the test generation procedure. The reason is that they serve no purpose while generating the tests. The presence of external outputs would enhance the

location of faulty cells. However, in the current integrated chip technology, fault location at the cell level - inside a chip - is not of much importance, since entire chips would be replaced anyway.

Finally the bound on the total number of tests is shown to be $O(n^4)$ and doesn't depend on the number of cells in the array except in cases where the length of the C-Test is greater than the length of the array. In both examples that were considered, the total number of C-Tests were much below the bound, suggesting that further analysis might tighten the bound considerably.

One way in which the bound can be further tightened is by using the concept of fault equivalence. We know that faults which have identical tests form an equivalence class [13]. Let there be $k_{ij}(i = 1,2,\ldots,mn; j \leq n-1;$ m=number of external inputs in the flow table; n=number of states in the flow table) fault equivalence classes corresponding to each (n-1) state faults with respect to a fault-free state transition. If $||k_{ij}||$ denotes the number of elements (state faults) in each j for a given i, then $1 \leq ||k_{ij}|| \leq (n-1)$.

Since all the elements in k_{ir} $(r \varepsilon j)$ have the same C-Test, our earlier bound on the number of tests needed to test a one-dimensional array (section 3.7) can be modified by taking the above fact into consideration. If there are

k_{ij} fault equivalence classes corresponding to every fault-free tansition then the total number of state faults to be considered for purposes of test generation is $\sum_{i=1}^{mn} \Sigma k_{ij}$. As a result of this the total number of tests needed (assuming a C-Test exists for every equivalence class k_{ir}) will be equal to $(\sum_i \sum_j k_{ij})(n(n-1))$. For $j<(n-1)$, $(\sum_i \sum_j k_{ij})(n(n-1)) < mn(n-1)(n-1)n$, the bound calculated in section 3.7.

An useful tool in the analysis of algorithms is the measure of complexity of the algorithm [1]. The measure of complexity can be broadly defined as the number of steps needed to arrive at the solution for a given algorithm. If the complexity is taken as the maximum complexity, then it is called the worst-case complexity. On the other hand if it is the average complexity, then it is called the expected complexity. Since it is difficult to make assumptions about the nature of the problem prior to its solution, the expected complexity will be more difficult to compute than the worst-case complexity. In order to study the complexity of the algorithm developed in section 3.8, we will use the worst-case complexity as a measure of the algorithm's complexity.

The worst-case complexity can be expressed as the maximum number of steps inovlved in arriving at the final

solution, and, in our case it is the C-Test set. The following is a list of the number of steps involved in generating the C-Test set using the algorithm in section 3.8.

Step 1. Transition Matrix multiplication: Let T be a n x n matrix (n is the number of states in the flow table). To compute one row in T^2 the complexity is $O(n^2)$ and hence for n rows in T^2 it is $O(n^3)$. Since the algorithm requires the generation of T matrices up to $(n-1)^{th}$ power, the number of steps needed is $(n-2)n^3$ and hence the worst-case complexity for the step is $O(n^4)$.

Step 2. Table Look-up for each NFP group: Corresponding to each NFP group we get the entries from all (n-1) columns in Table 2. Since there are (n-1) NFP pairs in each NFP group the total number of look-ups needed is mn(n-1)(n-1) and hence the complexity is $O(n^3)$.

Step 3. Sorting for each NFP group: Corresponding to each NFP group, each member in the NFP is sorted into a group L_r (refer to algorithm for notation) such that all the members in L_r have identical entries. The number of steps involved here is mn(n-1) and the complexity is $O(n^2)$.

Step 4. Determination of transitions: Corresponding to step 2 of the algorithm we have to determine the transition $Lq_r^k \xrightarrow{I_d} L_r$. Since $r \leq (n-1)$, the complexity for this step is $O(n^2)$.

Step 5. This corresponds to step 3 of the algorithm.

It involves mn(n-1) comparisons at best, and the complexity will be $O(n^2)$.

Step 6. If $\ell > n$, we should proceed to part (2) of the algorithm. If k NFPs corresponding to each fault-free transition do not have C-Tests of length $\ell \leq n$, then we have to make use of part (2) of the algorithm to generate C-Tests for (mnk) NFPs. For each k the total number of table look-ups needed in part (2) is (n-1)(n-1)n. Hence for (mnk) NFPs the complexity at this step is $O(n^4)$.

Step 7. As in step 3, the complexity of (mnk) sortings (as in Figure 28) is $O(n^2)$.

Step 8. The determination of transitions (as in Figure 28) requires (n-1) steps and the complexity is $O(n^2)$.

Step 9. Finally the comparisons (step 4, algorithm part (2)) has a complexity of $O(n^2)$.

From the above, the total worst-case complexity of the algorithm is $C = O(n^4) + O(n^3) + O(n^2)$. It can be seen that C increases exponentially with n but not as rapidly if $C = O(f(2^n))$. The complexity increases when $\ell > n$ for some NFPs and the worst-case corresponds to when $\ell > n$ for all NFPs. The following example illustrates the case when five NFPs have C-Tests of length $\ell > n$.

Example:

Design a C-Test set for the following flow table

implemented in a one-dimensional iterative array.

Flow Table:

	0	1
A	B	A
B	C	B
C	D	D
D	C	A

Using the algorithm in section 3.8 the following C-Test set was generated for all the NFPs.

C-Test Set:

(A,0001),(A,011001),(A,11),(A,1011011)

(B,0010), (B11)

(C,00),(C,0100),(C,1000),(C,110110)

(D,00),(D,0100),(D,1000),(D,101101).

In the above test set, the tests (A,1011011),(C,110110), (A,011001),(D,101101) have $\ell > n$, i.e., the length of the C-Test in each case is greater than four. The tests (A,0001),(A,1011011),(B,11),(B,0010),(C,00),(C,1000), (D,00),(D,1000) are capable of testing for more than one faulty transition. This results in lesser number of tests (due to fault equivalence). Using the theoretical bound calculated earlier the total number of C-Tests required is $(\Sigma\Sigma k_{ij})(n(n-1)) = (2+2+1+1+2+2+2+2)(4)(3) = 168$. For the above flow table the number of C-Tests obtained in

practice is 58. This is closer to 168 than 288, the bound calculated without taking into account the concept of fault equivalence. For the examples given in section 3.9 the total number of tests required were 22 and 38 which are far less than 288.

Finally it should be mentioned that fault equivalence (based on C-Tests) cannot be determined a priori, as existence of C-Tests cannot be determined a priori, (Theorem 3.2).

CHAPTER IV

TWO-DIMENSIONAL ARRAYS

4.1: Introduction

Over the past decade researchers (Unger [44], Minnick [31], Nicoud [33], Cappa [6]), have utilized two-dimensional iterative array realizations for the solution of specific problems such as multiplication, radix conversions, etc. Maintaining such arrays fault-free becomes imperative, and many like Kautz [17], Friedman [12], Prasad [35], Seth [38] and Landgraff [21] have considered the problem of fault diagnosis in such arrays. All of them with the exception of Prasad were concerned with single cell fault occurrences only. In this chapter we will investigate both single as well as multiple-cell fault occurrences in two-dimensional iterative arrays.

In a two-dimensional array an input to an arbitrary cell deep inside the array is not generated by a single cascade of cells to the left of the cell under consideration, but by an entire sub-array of cells which provides both the X and Y states to the arbitrary cell. Hence we cannot determine a priori whether a particular input state can be applied to a cell deep inside the array. In order to facilitate the discussion of conditions necessary for the existence of such a set of inputs to all the cells in the array, we should briefly comment on the tessellation

problem.

A tessellation is defined as a complete set of cell input states which are compatible with one another along cell inter-faces, along the entire array, (Kautz [17]). In other words tessellation corresponds to a cycle in a pair graph where each vertex is a state pair. In order that an arbitrary cell can be cycled through all possible input combinations it is necessary that one or more tessallations exist such that they cover all the input combinations applicable to a cell. As said earlier, existence of such a set of tessellations cannot be determined a priori. We discussed in Chapter II some of the methods utilized to arrive at a set of tessellations.

Kautz [17] states that if no two entries in a column in the flow table of a cell corresponding to a two-dimensional array are the same, then the cell can be tested with a minimum number of tests 'mn' where 'm' is the number of columns and 'n' is the number of rows in the flow table. The tessellation set is built as follows. An arbitrary input state is applied to all the cells in the diagonal file of an array. The outputs from the file will be the inputs to the adjoining file and all the inputs are the same. Since the number of input states is finite, the input state at the diagonal file D_1 will repeat at a finite diagonal file D_k and finally the set of 'k' diagonal files is iterated over the entire two-dimensional

plane. We see now, that all the inputs in the diagonal file can be applied to all the cells in a two-dimensional array. Cell states that are not applied in this stage can be used in successive tessellation sets (if they can be) using the same technique suggested above. Figure 32 illustrates this technique.

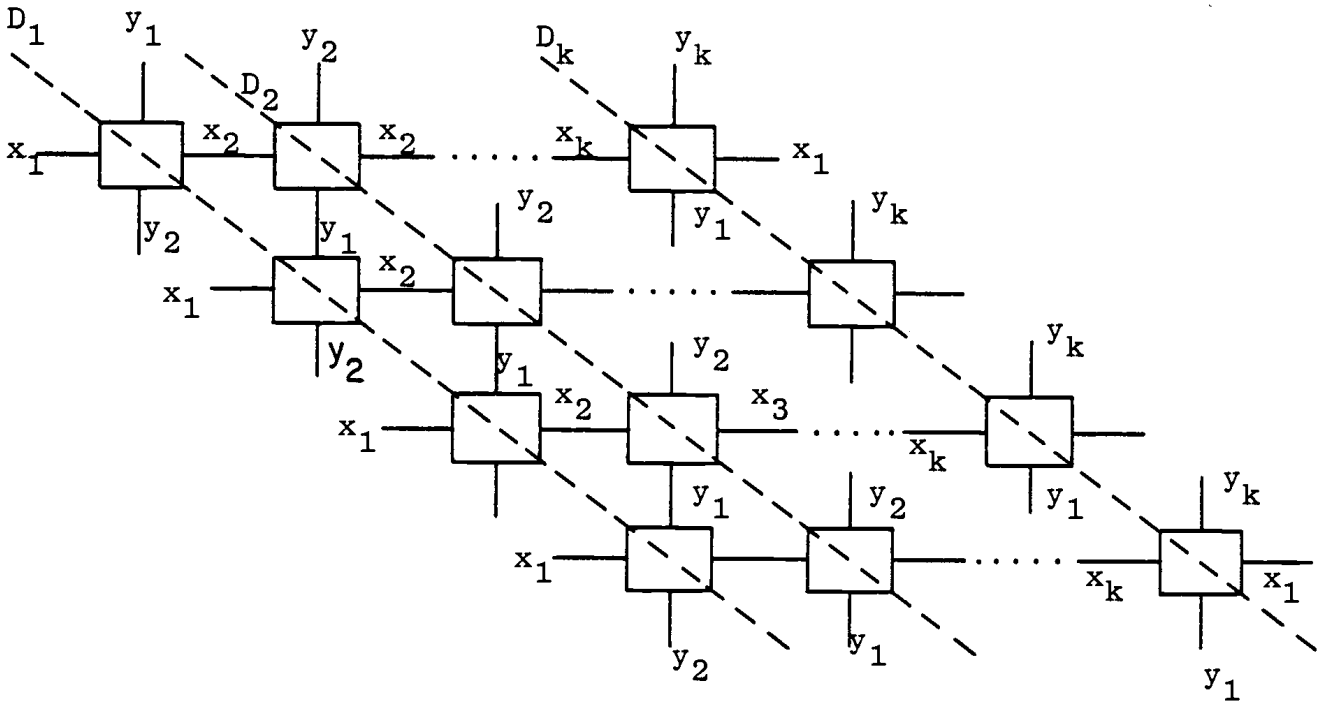


Figure 32. Diagonal construction of tessellation.

The tessellation set for the above figure, Figure 32 is $(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k)$.

We shall, however, approach the problem of test generation in two-dimensional arrays from a different view point. The two-dimensional array is visualized as a one-dimensional array where each cell is viewed as a column or row of cells in a two-dimensional array combined into a

single cell. I.e., either vertically or horizontally combined into a single cell. We will call the former vertical compression and the latter horizontal compression. Each cell in the equivalent one-dimensional array has 'n' horizontal inputs and one vertical input with respect to the horizontal compression, and one horizontal input and 'n' vertical input with respect to the vertical compression.

4.2: Model for a Two-Dimensional Array

Thus we visualize the two-dimensional array given in Figure 10 as an equivalent one-dimensional array compressed either horizontally or vertically. Figure 33 represents a two-dimensional array compressed horizontally. Each module in the figure contains 'n' cells corresponding to that column which contains the module. Similarly we can get a one-dimensional array that is equivalent to a two-dimensional array compressed vertically as in Figure 34.

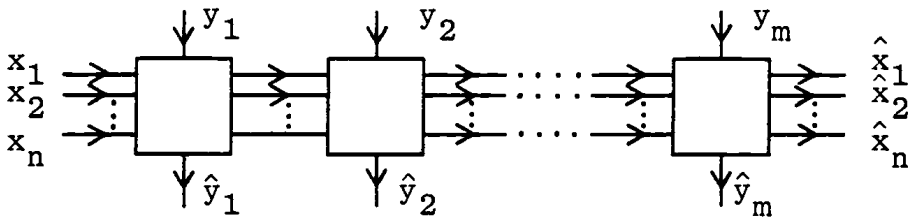


Figure 33. Horizontal compression of a two-dimensional array into a one-dimensional array.

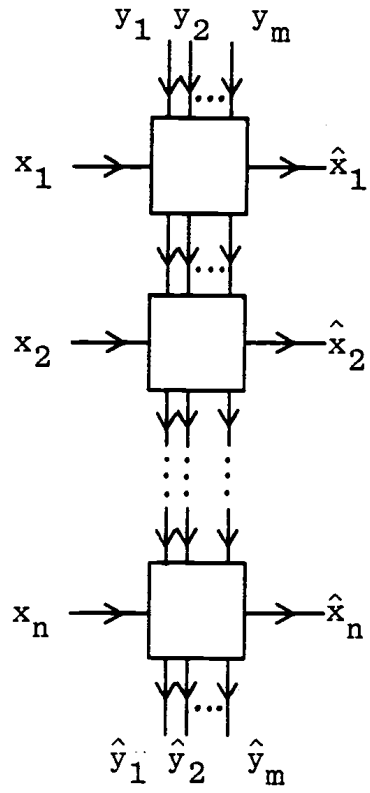


Figure 34. Vertical compression.

4.3: Fault Propagation in Two-Dimensional Arrays

We assume that the nature of faults in a two-dimensional array will be the same as in one-dimensional case, i.e., State Faults. The faulty behavior of the 'ij'th cell in the two-dimensional array can be studied under two conditions.

Case (a) : $\hat{x}$ output is faulty

Case (b) : $\hat{y}$ output is faulty

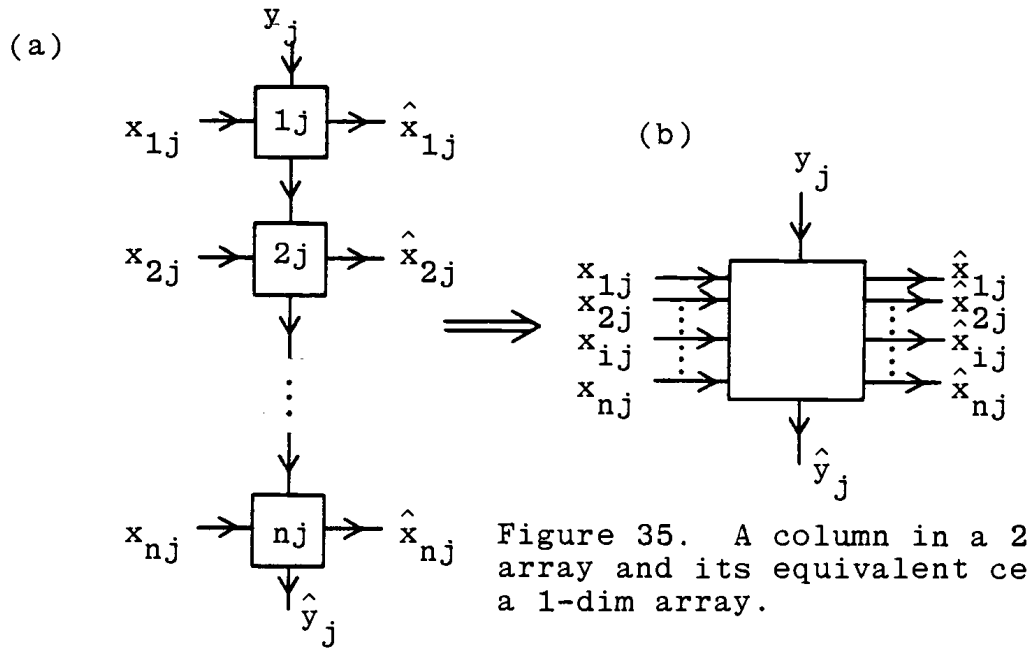


Figure 35. A column in a 2-dim array and its equivalent cell in a 1-dim array.

Let the $\hat{x}$ output $\hat{x}_{ij}$ of the ij^{th} cell in the two-dimensional array (Figure 35(a)) be faulty. In the corresponding one-dimensional model (Figure 35(b)) the same fault corresponds to a fault in the output state $(\hat{x}_{ij}\hat{x}_{2j}\ldots\hat{x}_{ij}\ldots\hat{x}_{nj})$ of the j^{th} cell module. If the flow table corresponding to the equivalent one-dimensional array is reduced we know from the previous chapter that a state fault can be easily propagated to an external output. I.e., there always exists a set of inputs that will propagate the effect of fault to an external output from the site of fault (Lemma 3.1). Hence in the above case the $\hat{x}$ fault can be propagated in the horizontal direction.

Case (b):

In order to propagate the faulty $\hat{y}$ output of the ij^{th} cell in the horizontal direction, it is necessary that the following condition is satisfied. For the $\hat{y}_{ij}$ to

cause a change in the $\hat{x}$ output of the $((i+1)j)^{th}$ cell, it is necessary that no two columns are alike in at least one of the rows of the flow table corresponding to the $\hat{x}$ output (Kautz [17]). Under such condition a fault $\hat{y}_{ij}$ will be propagated to $\hat{x}_{(i+1)j}$. The propagation condition now reduces to case (a) and $\hat{y}_{ij}$ fault can be propagated in the horizontal direction if the aforesaid conditions are satisfied.

Example: $\hat{x} = x \oplus y$

$\hat{y} = x \equiv y$

Flow Table:

$\hat{x}$:

x \ y	0	1
0	0	1
1	1	0

$\hat{y}$:

x \ y	0	1
0	1	0
1	0	1

In a two-dimensional array realizing the above $\hat{x}$ and $\hat{y}$ functions, any fault in the $\hat{y}$ output will cause the change in the $\hat{x}$ output of the cell below the faulty cell. This is due to the fact that both the columns in the $\hat{x}$ flow table are different for both the rows, and any change in $\hat{y}$ will cause a change in the $\hat{x}$ output.

Finally when both $\hat{x}_{ij}$ and $\hat{y}_{ij}$ are faulty, the fault can still be propagated in the horizontal direction, provided the equivalent one-dimensional-array-flow-table is

is reduced and the $\hat{x}$ flow table has no two columns alike. In this way the $\hat{y}_{ij}$ fault will cause a change in the $\hat{x}_{(i+1)j}$ output and the overall effect will be equivalent to a fault in the $\hat{x}$ leads.

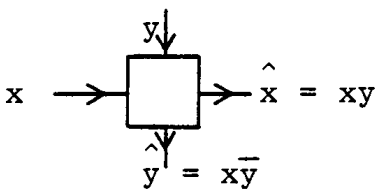
Propagation is also possible in the vertical direction in all the above cases provided the flow table condition in case (b) is changed such that no two rows are alike in at least one of the clumns of the $\hat{y}$ flow table. It is now evident that the multiple-fault Theorem 3.1 in Chapter III will be readily applicable to the two-dimensional array also.

4.4: Test Derivation Procedure

In order to generate tests for the equivalent one-dimensional array we have to develop the flow table corresponding to a typical cell in the equivalent one-dimensional array. A typical cell in the two-dimensional array is represented by two flow tables, one for the $\hat{x}$ output and the other for the $\hat{y}$ output. The size of the new flow table representing a cell in the equivalent one-dimensional array will depend on the physical dimension of the two-dimensional array. For a $n \times m$ two dimensional array the equivalent one-dimensional array cell will have a flow table of size $2^n \times 2$, the 2^n corresponding to the rows and 2 to the columns. This new flow table can be obtained using the algebraic expressions for $\hat{x}$ and $\hat{y}$.

Once we have the flow table, the rest of the procedure is the same as that of one-dimensional case. The method involves the generation of T-matrices from the flow table, and later, using Part (1) and (2) of the algorithm in section 3.8 to generate the necessary C-Tests.

However there are certain variations as compared to the one-dimensional array case. In the one-dimensional array it was assumed that the flow table is strongly connected. But in a two-dimensional case such an assumption is not always necessary. This can be explained as follows. In a two-dimensional array it is our wish to apply all possible input combinations to all cells in the array, but this is not always possible. As an example consider the following cell realized in a two-dimensional array.



$\hat{x}$:

x \ y	0	1
0	0	0
1	0	1

$\hat{y}$:

x \ y	0	1
0	0	1
1	0	0

$(\hat{x},\hat{y})$:

x \ y	0	1
0	(0,0)	(0,1)
1	(0,0)	(1,0)

The $(\hat{x},\hat{y})$ flow table is not strongly connected and the input combination (1,1) cannot be applied to an arbitrary cell inside the array. For example it cannot be applied to the lower right cell in a 2 x 2 array. However if the

states corresponding to the strongly-connected subgraphs of a weakly-connected state graph (of the equivalent one-dimensional array) cover all the input combinations applicable to a cell in the two-dimensional array, we can then test the two-dimensional array for all faults.

The criterion for the existence of a C-Test in the two-dimensional case is the same as in the one-dimensional case. If, in addition to the above covering condition, conditions stated in Theorem 3.2 are satisfied there will be a C-Test for all the state faults in the two-dimensional array. Whenever the length of the test is greater than the length of the equivalent one-dimensional array, the test can still be used, but in a truncated form. This will result in the number of tests being dependent on the dimension of the array.

To this point we have been discussing two-dimensional arrays compressed horizontally into an equivalent one-dimensional array. The same techniques can be used for arrays compressed vertically too.

4.5: Fault Location in Two-Dimensional Arrays

Whenever there is a single cell in a two-dimensional array, fault location is possible provided the conditions in case (a) and (b) of section 4.3 are satisfied.

If cell ij is faulty in Figure 36, a test applied to detect this fault will produce an output $(\hat{x}_1^f \hat{x}_2^f \ldots \hat{x}_i^f \ldots \hat{x}_n^f)$

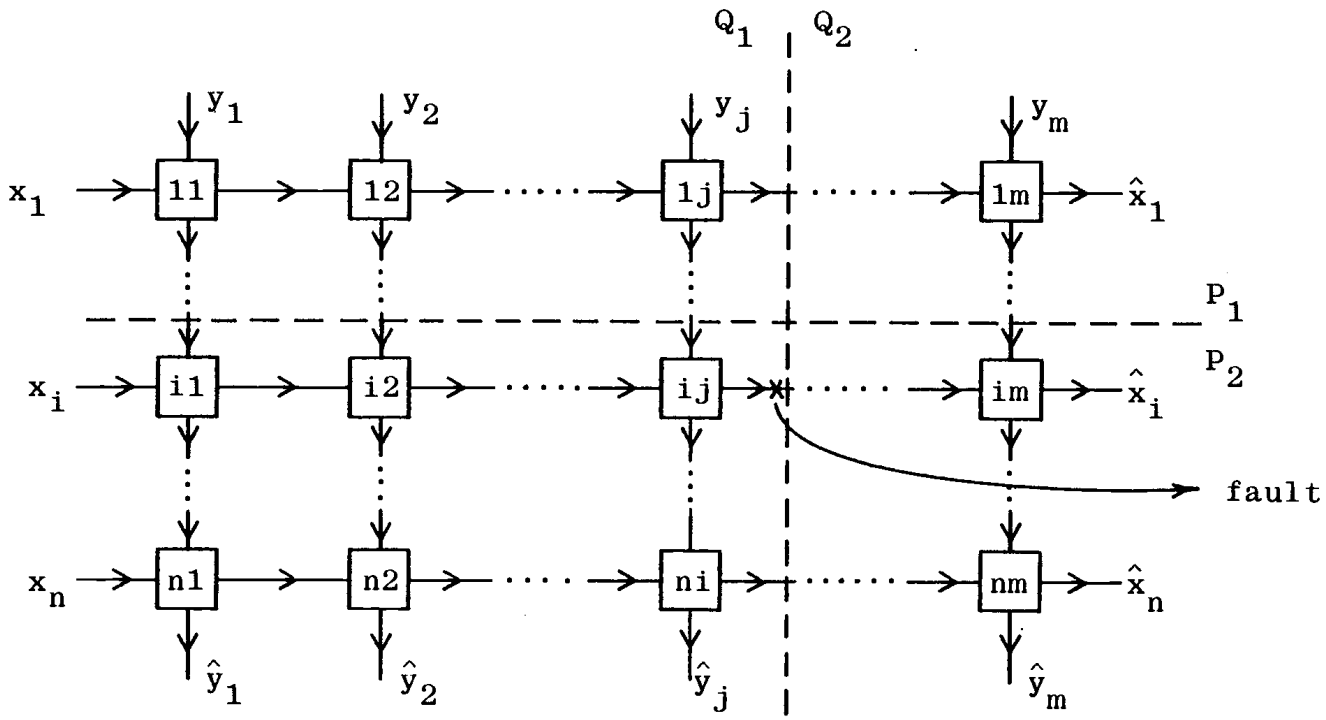


Figure 36. Two-dimensional array with a fault in the ij^{th} cell.

as opposed to the output under normal condition $(\hat{x}_1\hat{x}_2\ldots\hat{x}_i\ldots\hat{x}_n)$. Since the flow table condition in case (b) of section 4.3 is satisfied, there will be a change in $\hat{x}_r$ for $r \geq i$ and no change in $\hat{x}_k$, $\forall\, k<i$. Hence $\hat{x}_k = \hat{x}_k^f$, $\forall\, k<i$ and $\hat{x}_r \neq \hat{x}_r^f$, $r \geq i$. This will partition the array horizontally into P_1 and P_2. In addition to the condition in case (b), if no two rows are alike in at least one column of the flow table (of the $\hat{y}$ output), there will be a change in the $\hat{y}_r$, $r \geq (j+1)$ and no change in $\hat{y}_k$, $\forall\, k<(j+1)$. The array is once again partitioned into Q_1 and Q_2. The intersection of these two partitions determine the location of the faulty cell at most to the adjacent column and row.

Example:

This example illustrates fault location in two-dimensional array. Consider the two dimensional array realizing the cell functions shown in Figure 37. Assume cell '33' to be faulty and let the faulty transition be $1\ x\ 0 \longrightarrow 0$ $(x\ x\ y \longrightarrow \hat{x})$ as opposed to $1\ x\ 0 \longrightarrow 1$. Using the external inputs as shown we can apply the input combination to the faulty cell (33). Under normal conditions, the external outputs of the array $(\hat{x}_1\hat{x}_2\ldots\hat{x}_5)$ will be (11111). Because of the fault in $\hat{x}_{33}$, this is propagated horizontally to the external output $\hat{x}_3$ (since the flow

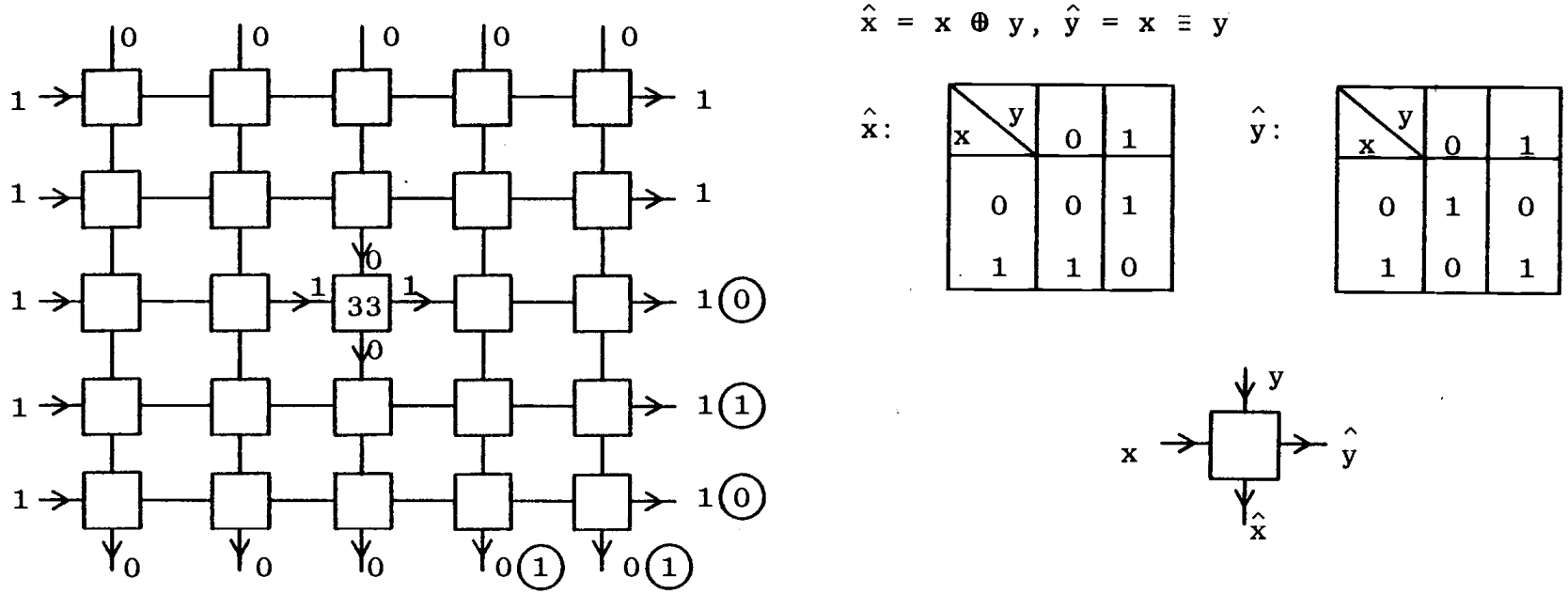


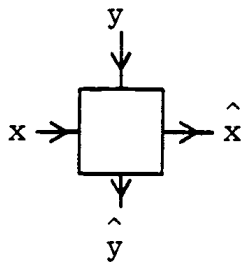
Figure 37. Two-dimensional array with a faulty cell (33)

table for $\hat{x}$ satisfies the condition in case (b)). Hence the horizontal external outputs of the array will change due to the presence of the faulty cell '33' as shown in Figure 37 by circled entries. The external output due to the fault is (11010). This partitions the array horizontally into two groups. I.e., (row 1, row 2);(row 3, row 4, row 5). Similarly the $\hat{y}$ flow table satisfies the condition in case (b) and the fault is propagated to the 4^{th} and 5^{th} column of the array, resulting in a change in the external vertical outputs of column 4 and 5, i.e., (00011). Once again the array is partitioned into two groups (columns 1, 2 and 3), (columns 4 and 5). Finally we find the intersection of these two partitions determines the location of the faulty cell as cell '33'.

4.6: Example for Two-Dimensional Array C-Test Generation

The algorithm discussed in Chapter III will be illustrated here with an example for a two-dimensional array: Realize a two-dimensional array that realizes the following cell functions and generate C-Tests whenever they exist.

$$\hat{x} = \bar{x} + y \ , \ \hat{y} = xy$$



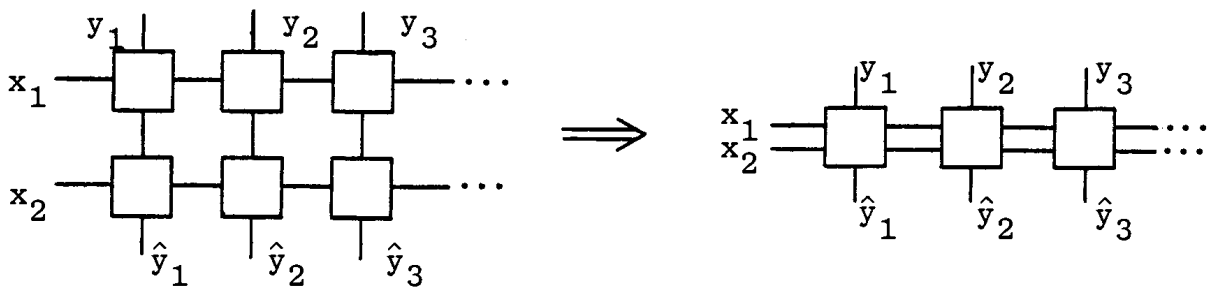
$\hat{x}$:

x \ y	0	1
0	1	1
1	0	1

$\hat{y}$:

x \ y	0	1
0	0	1
1	0	0

Construction of a flow table for a cell in the one-dimensional array equivalent to a (2xm) two-dimensional array:



State Assignment

x_1	x_2	
0	0	A
0	1	B
1	0	C
1	1	D

Flow Table

x \ y	0	1
A	D	D
B	C	D
C	B	D
D	A	B

$$T = \begin{vmatrix} \emptyset & \emptyset & \emptyset & 0,1 \\ \emptyset & \emptyset & 0 & 1 \\ \emptyset & 0 & \emptyset & 1 \\ 0 & \emptyset & 1 & \emptyset \end{vmatrix} \qquad T^2 = \begin{vmatrix} 0,1 & \emptyset & 2,3 & 0 \\ 2 & 0 & 3 & 1 \\ 2 & \emptyset & 0,3 & 1 \\ \emptyset & 2 & \emptyset & 0,1,3 \end{vmatrix}$$

$$T^3 = \begin{vmatrix} \emptyset & 2,6 & \emptyset & 0,1,3 \\ & & & 4,5,7 \\ 2 & 0 & 0,3 & 1,4,5,7 \\ 2 & 0,6 & 3 & 1,4,5,7 \\ 0,1,6 & \emptyset & 2,3,4,7 & 5 \end{vmatrix}$$

Using part (1) and (2) of the algorithm in Chapter III the C-Tests were generated for 2 x 2 array

(A,00);(B,00);(C,00);(D,00);(C,11)

C-Tests do not exist for faulty transitions from A, B and D on an input 1.

Check: The test sets (B,00), (C,00) will be sufficient to apply the input combinations (0,0) and (1,0) to every cell in a 2 x 2 array. Figure 38 shows how the above test sets can be applied to a 2 x 2 array.

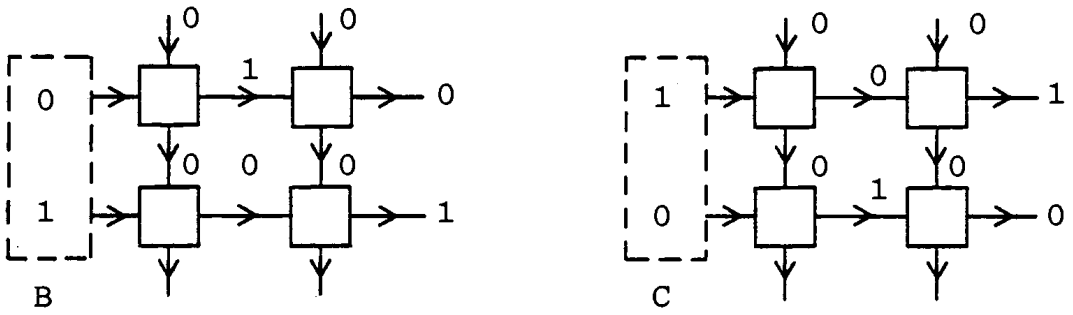
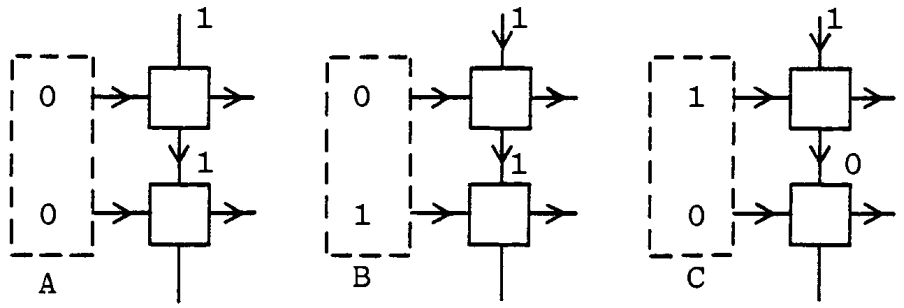


Figure 38. Application of (B,00) and (C,00) to a 2x2 array.

However the input combinations (0,1) and (1,0) are not covered by any of the C-Tests. It can be seen from the equivalent flow table that the transitions

A $\xrightarrow{1}$, B $\xrightarrow{1}$, C $\xrightarrow{1}$ covers the above two input combinations.

i.e.,



Using (C,11) the transition $C \xrightarrow{1}$ can be applied to all the cells in a $2 \times m$ array and hence a 2×2 array. Since we do not have C-Tests for $A \xrightarrow{1}$ and $B \xrightarrow{1}$ the number of tests needed to apply these input combinations to all the cells in the two dimensional array will depend on the dimension of the array. We need (B,11), and (C,01) to apply $B \xrightarrow{1}$ to each cell in the equivalent one-dimensional array (all the cells in a two-dimensional array of size 2×2). Similarly we need (A,10), (D,01) to apply $A \xrightarrow{1}$ to the 2×2 array.

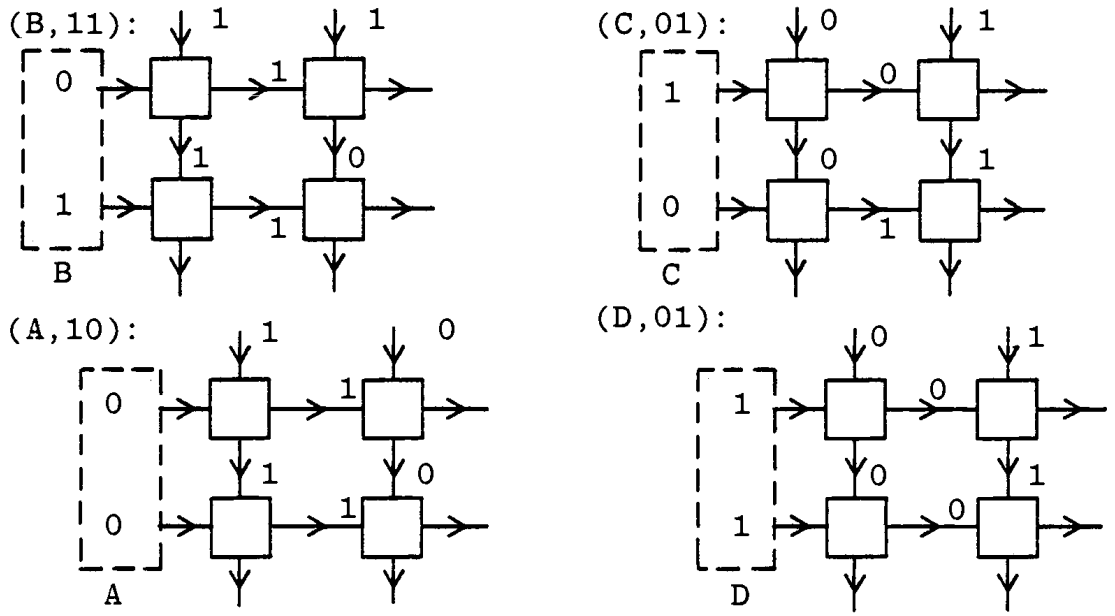


Figure 39. Application of (B,1), (C,01), (A,10) and (D,01) to a 2×2 array.

Hence the number of tests needed to completely test the 2x2 array is seven tests. We have another example where there are C-Tests for all the faulty transitions. A two-dimensional array realizing the functions $\hat{x} = x \oplus y$ and $\hat{y} = x \equiv y$ can be tested for all faults using the

following C-Tests. It can be verified using the algorithm part (1) that the C-Test corresponds to (A,00), (B,00), (C,00), (D,00), (A,1111), (B,1111), (C,1111), (D,1111). The test set is for a 2 x m array.

Although analyzing the two-dimensional array (for fault detection) in terms of its equivalent one-dimensional array poses certain computational problems, it can be seen that general procedures derived for a regular one-dimensional array can be directly extended to the two-dimensional iterative array.

CHAPTER V

SUMMARY AND CONCLUSION

In this dissertation we have developed an algorithm capable of generating a set of concurrent tests called C-Tests that can detect all faults in combinational iterative arrays. The fault class has been assumed to be state faults. These encompass a wider class of faults compared with the familiar "stuck-at" type faults that are highly dependent on the hardware realization of the circuit. The tests have been derived under the assumption that both single or multiple cells can be faulty in the array.

Using a transition matrix of the flow table, we generate matrices up to the $(n-1)^{th}$ power. By comparing rows in the T^k matrix, corresponding to the faulty and fault-free transition, we arrive at test sequences called C-Tests. Such tests are capable of detecting faults in more than one cell simultaneously. It has been shown that such tests do not exist for all state faults and a graph-theoretic condition has been imposed on the flow graph for the existence of such tests. We have also suggested a modification to the cell such that the modified cell can be tested completely using only C-Tests.

Earlier work (Kautz [17], Prasad [35], Landgraff [21]) has mainly concentrated on the derivation of tests for a particular class of flow tables, such as ones that

have permutation columns. In our case we have treated arbitrary flow tables that possess no such special property.

In the case of two-dimensional iterative arrays, it has been converted into an equivalent one-dimensional array. This is done by compressing each row or column of the two-dimensional array into a single cell in the equivalent one-dimensional array. It is found that results derived for one-dimensional array are equally applicable to the two-dimensional array. This concept makes it easier to analyze the two-dimensional array for faults and the test generation algorithm for the one-dimensional array is directly applied to the equivalent one-dimensional array. Although this method is computationally complex, it shows how a two-dimensional array can be viewed as an equivalent one-dimensional iterative array. This is in contrast to the past work which concerned itself mainly with the tessellation problem. We have also shown that fault location is possible in two-dimensional iterative arrays when certain flow table conditions are satisfied.

In conclusion, the method proposed in this work is original in nature, in that it treats the general case of iterative array fault detection. However we cannot avoid the necessary fact that the computational complexity of test derivation procedure increases exponentially with the magnitude of 'n'. Future work should aim at classify-

ing flow tables that satisfy the graph theoretic condition. This may lead to design modifications to the cell prior to the test generation procedure, thereby making the cells easier to test with a lesser number of tests.

BIBLIOGRAPHY

1. Aho, A. V., et al., "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., Ed. 1976.

2. Akers, S. J., "A Rectangular Logic Array," IEETC, Vol. c-21, pp. 848, August 1972.

3. Armstrong, D. B. "On finding a nearly minimal set of fault detection tests for combinational networks," IEETC, Vol. c-15, pp. 66-73, Jan. 1966.

4. Avizienis, A., "Fault tolerant systems," IEETC, c-25, pp. 1304-1312, Dec. 1976.

5. Bossen, D. C. and S. J. Hong, "Cause effect analysis for multiple fault detection in combinational networks," IEETC, c-20, pp. 1254-1261, Nov. 1971.

6. Cappa, M. and V. C. Hamacher, "An augmented iterative array for high speed binary division," IEETC, Vol. c-22, pp. 172-76, Feb. 1973.

7. Chia, H. S. and C. L. Coates, "Tessallation aspect of combinational cellular array testing," IEETC, Vol. c-23, pp. 363-68, Apr. 1974.

8. Deo, R. S. "Graph theory with applications to engineering and computer science," Prentice Hall, Inc., Englewood Cliffs, N. J., Ed. 1974.

9. Dias, F. J. O. "Truth table verification of iterative logic array," IEETC, Vol. c-25, pp. 605-13, June 1976.

10. Eldred, R. D. "Test routines based on symbolic logical statements," Journal of Assn. for Computing Machinery, Vol. 6, pp. 33-36, Jan. 1959.

11. Friedman, A. D. "Fault detection in redundant circuits," IEETC, Vol. c-22, pp. 835-39, Sept. 1973.

12. Friedman, A. D. and P. R. Menon, "Fault detection in iterative logic array," IEETC, Vol. c-20, pp. 524-35, May 1971.

13. Friedman, A. D. and P. R. Menon, Fault detection in digital circuits," Prentice-Hall Series, N. J.; Ed. 1971.

14. Givone, Roessi, "Multidimensional iterative circuits-General properties," IEETC, Vol. c-21, pp. 1067-79, Oct. 1972.

15. Hennie, F. C., "Iterative arrays of logical circuits," M.I.T. Press, Cambridge, Mass. Ed. 1967.

16. Jump, J. R. and D. R. Fritsche, "Microprogrammed array," IEETC, Vol. c-21, pp. 974-84, Sept. 1972.

17. Kautz, W. H. "Tests for faults in combinational cellular logic arrays," Proc. Fifth Ann. Symp. on Switch. and Aut. Theory, pp. 161-74, 1967.

18. Kautz, W. H. "Cellular logic-in-memory arrays," IEETC, Vol. c-18, pp. 719-27, Aug. 1969.

19. Kautz, W. H. "Cellular interconnection arrays," IEETC, Vol. c-17, pp. 443-51, May 1968.

20. Kohavi, Z. "Designing set of fault detection tests for combinational circuits," IEETC, Vol. c-20, pp. 1463-69, Dec. 1971.

21. Landgraff, R. W. and S. S. Yau, "Design of diagnosable iterative array," IEETC, Vol. c-20, pp. 867-77, Aug. 1971.

22. Levitt, K. N. and W. H. Kautz, "Cellular arrays for the parallel implementation of binary error correcting codes," IEETC on Inf. Theory, Vol. 15, pp. 597-607, Sept. 1969.

23. Maitra, K. K., "Cascaded networks of two input flexible cells," IEETC, Vol. c-11, pp. 136-43, Apr. 1962.

24. Maruoka, A., "Logical network of flexible cells," IEETC, Vol. c-22, pp. 347-58, Apr. 1973.

25. McCluskey, E. J., "Iterative combinational switching networks-general design consideration," IRE Trans. Comp., Vol. EC-7, pp. 285-91, Dec. 1958.

26. McCluskey, E. J. and J. F. Poage, "Derivation of optimum test sequences for sequential machines," Proc. 5th Ann. Symp. on Switch. Th. and Logic Design, pp. 121-32, 1964.

27. Mei, K. C. Y., "Bridging stuck-at-faults," IEETC, Vol. c-23, pp. 720-27, July 1974.

28. Metze, G. and C. Cha, "Multiple fault diagnosis in combinational networks," Proc. 12th Allerton Conf. on Cir. and Sys. Theory, pp. 244-52, Oct. 1974.

29. Metze, G. and D. R. Schertz, "Design of multiple fault diagnosable network," IEETC, Vol. c-20, pp. 1361-64, Nov. 1971.

30. Minnick, R. C., "Cutpoint cellular logic," IEETC, Vol. 13, pp. 685-98, Dec. 1964.

31. Minnick, R. C., "Cobweb cellular array," Proc. FJCC, Vol. 27, pp. 327-41, 1965.

32. Mukopadhyay, A., "Unate cellular logic," IEETC, Vol. c-18, pp. 114-21, Jan. 1969.

33. Nicoud, J. D.,"Iterative arrays for radix conversion," IEETC, Vol. c-20, pp. 1479-89, Dec. 1971.

34. Poage, J. F., "Derivation of optimum tests to detect faults in combinational circuits," Proc. Symp. on Math. Th. of Automata., Poly. Inst. Brooklyn, pp. 483-528, 1963.

35. Prasad, B. A. and F. G. Gray, "Multiple fault detection in arrays of combinational cells," IEETC, Vol. c-24, pp. 794-892, Aug. 1975.

36. Roth, J. P., "Diagnosis of automata failure: a calculus and a method," IBM Journal of R & D., Vol. 10, pp. 278-91, 1966.

37. Sellers, F. F. and M. A. Hsiao, "Analyzing errors with boolean difference," IEETC, Vol. c-17, pp. 676-83, July 1968.

38. Seth, S. C., "Fault diagnosis of combinational cellular array," Proc. 7th Ann. Conf. on Cir. and Sys. Th. Urbana, Ill.,pp. 272-83, Oct. 1969.

39. Seshu, S., "Transition matrices of sequential machines," IRE Trans. on Elec. Computers, Vol. EC-6, pp. 276-85, Dec. 1957.

40. Short, R. A., "The attainment of reliable digital system through the use of redundancy-A survey," Computer Group News, pp. 2-17, March 1968.

41. Short, R. A., "Two rail cellular cascades," AFIPS, FJCC, No. 27, pp. 355-70, 1965.

42. Tammaru, E., "Efficient testing of combinational logic cells in large scale arrays," IEEE Computer Repository, R. 69-115, IEEE, N.Y., 1967.

43. Thurber, K, J., "Fault location in cellular arrays," Proc. FJCC, Vol. 35, pp. 81-88, 1969.

44. Unger, S. H., "Pattern recognition using two-dimensional bilateral iterative combinational switching circuits," Proc. of the Symp. on Math. Th. of Automata, pp. 577-91, April 1962.

45. Wang, H., "Dominoes and the AEA case of the decision problem," Proc. of the Symp. on Math. Th. of Automata, Poly. Inst. Brooklyn, Vol. 12, pp. 23-25, April 1962.

46. Yau, S. S. and Y. Tang, "An efficient algorithm for generating complete test sets for combinational logic circuits," IEETC, Vol. c-20, pp. 1245-51, Nov. 1971.