THE DESIGN OF C-TESTABLE ARITHMETIC UNITS

By

Sin-Min Chang

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering and Systems Science

ABSTRACT

THE DESIGN OF
C-TESTABLE ARITHMETIC UNITS

By

Sin-Min Chang

Due to the regular and iterative structure of iterative logic arrays (ILAs), this thesis presents the C-testable designs that can be tested with a test set of constant length regardless of the circuit size. The concept of C-testability developed for ILAs is applied to the design of C-testable array multipliers and dividers. The results show that the proposed design of n-by-n C-testable multipliers can be fully tested with 16 test patterns, while the n-by-n restoring and nonrestoring array dividers can be tested with 40 and 20 test patterns respectively. Algorithms that generate the test patterns and expected outputs are also provided.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

I. Introduction

Rapid advances in semiconductor fabrication technology have made possible the implementation of digital circuits with a very large number of devices on a single chip. The complexity is coupled with an increase in the ratio of logic to pins which drastically reduces the controllability and observability of the logic on the chip [1]. As a result, testing of such high-complexity circuits is very difficult. One of the important issues associated with circuit testing is fault detection. In general, fault detection is carried out by applying a sequence of test inputs and observing the resulting outputs. The major cost of testing includes the generation of test sequences and their application. To reduce the cost of testing, it is necessary to minimize the length of the test sequence [2].

An *Iterative logic array* (ILA) consists of several identical cells with identical interconnections between cells. Due to its regular and iterative structure, designs of C-testable ILAs that can be examined with a test set of constant length irrespective of the circuit size, have been presented [3-5]. Recently, array multipliers of reasonable size have been implemented on a single VLSI (Very Large Scale Integrated) circuit chip [6,7]. The concept of C-testability has been applied to the design of C-testable array multipliers [8].

The aim of this thesis is to present the designs of C-testable arithmetic units, such as array multipliers and array dividers, and their test generation procedures. In the next chapter, the testing of ILAs and previous work related to the C-testable designs are discussed. The inherent drawbacks in the previous work are also pointed out. In Chapter III and IV, the design and test generation of C-testable array multipliers and dividers are proposed. Finally, the conclusions and future research directions are given in Chapter V.

II. Background

2.1. The Testing of Iterative Logic Array

An *iterative logic array* (ILA) consists of several identical cells with identical interconnections between cells. This type of circuit configuration offers the advantages of structural regularity, like, ease of circuit and logic design, ease of placement and routing [10]. In a 1-D ILA, the cells are organized in a row, such as ripple-carry adders, while in a 2-D ILA the cells are organized in a matrix of rows and columns, such as array multipliers. In each direction of signal flow, more than one signal line is allowed.

As the complexity of the VLSI system increases, the reliability issue becomes more important than ever before. The method to make sure that a combinational circuit is functionally correct is to apply all the possible inputs and examine the corresponding output signals. However, it is impossible to do so on a large system. For example, if the number of input lines is 32, then the number of test patterns to detect the permanent faults is 2^{32}. This requires too much time for testing and too much memory space to store the test patterns. Moreover, the ILAs have the characteristics of unlimited expansion making the testing of ILAs highly interesting.

The test procedure is to apply test patterns to the accessible input terminals of the array, referred to as *primary inputs*, and to observe the results at the accessible output terminals, refered to as *primary outputs*. The observed results are verified by comparing them with the expected results. These accessible terminals are usually the boundaries of the array.

Faults in an ILA may occur either in the intercell connection, or in the array cells. The former is covered by either the input or output fault of the corresponding cell; the latter assumes that the faulty cell can change its truth table permanently in any arbitrary way as long as it remains a combinational circuit. However, it is assumed that there is no bridging fault between cells [8]. In practice, there are two fault-models at the array level [10]: *Single Cell Fault Model* (SCFM) and *Multiple Cell Fault Model* (MCFM). The former indicates only one cell out of the whole array can be faulty, and the latter means an arbitrary number of cells can be faulty.

Basically, an ILA can be tested exhaustively using the truth table of a whole array. The size of the test set is exponential to the number of cells. In recent years, two categories of ILAs that simplify their testing have been studied: *C-testable* and *L-testable* ILAs. The former is an ILA which can be tested with a constant test size irrespective of the number of cells in the ILA, and the latter is an ILA that requires a test size linear to the number of cells.

2.1.1. L-testable ILAs.

The test problems of ILAs under SCFM were first studied by Kautz [3]. Consider a 1-D ILA, as shown in Figure 1. Each cell receives an input x from its left-hand neighbor and an external input z. It generates an external output $\hat{z}$ and transmits an output $\hat{x}$ to its right-hand neighbor. The controllable inputs of the array consist of the x-input to the leftmost cell and the z-inputs to all cells. All $\hat{z}$-outputs and the $\hat{x}$-output of the rightmost cell are observable. We assume that the z-input of cell i, or z_i, is independent to the z-input of cell j, or z_j, for $i \neq j$. Kautz [3] characterized the following necessary and sufficient conditions for L-testibility of a general ILA under SCFM.

Condition 1: A complete set of test must be applied to the input terminals of any cells in the array.

Condition 2: For each test, any effect of the fault must be propagated to an observable output.

More specifically, consider the 1-D ILA of Figure 2. Each cell receives three inputs, x_i , z_i and, z_i', and produces an output, $\hat{W}_i$. Suppose the cells are connected in such a way that the output of a cell is fed to its right as shown. Since the only primary output of such an ILA is the output of the rightmost cell, a fault may not be detected unless it can be propagated to the primary output.

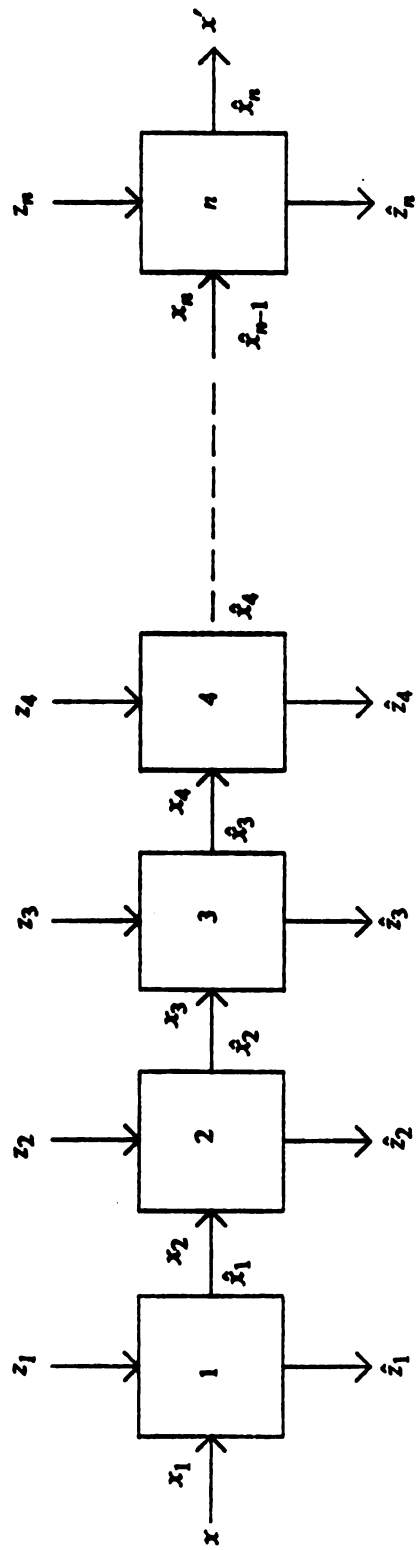


Figure 1. 1-D ILA with primary outputs $\hat{Z}_i$ and x'.

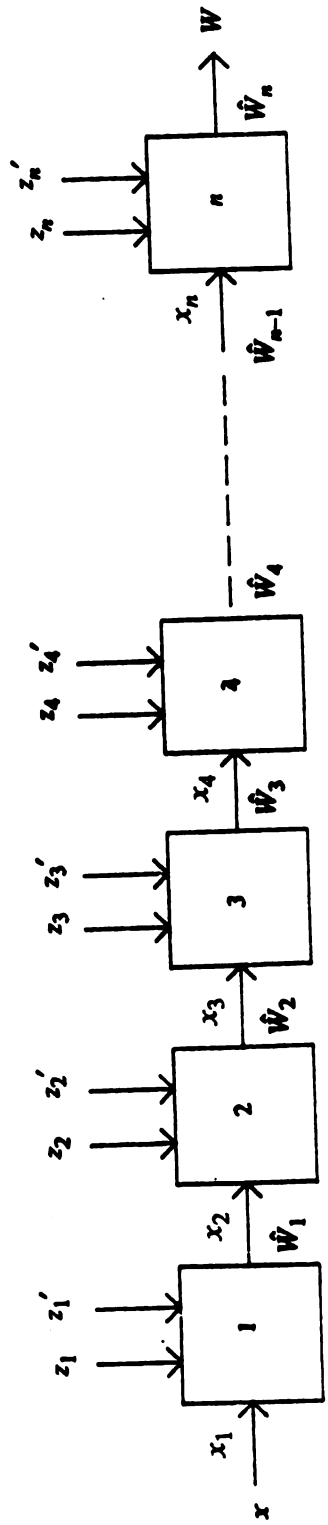


Figure 2. 1-D ILA with a primary output W.

In general, the cell behavior can be described by a truth table. For example, Table 1(a), which is the carry output of a full adder, describes the cells in Figure 2. Table 1(a) consists of columns for input x_i and rows for both z_i and z_i'. Each entry represents the output of the cell operation. For notational simplicity, we denote $\hat{W}_i=\hat{W}(x_i,z_i,z_i')$, where $x_i = \hat{W}_{i-1}$, i=1, 3, ..., n, $x=\hat{W}_0$, and $\hat{W}_n=W$. Suppose there is a faulty cell in this array and its function is changed so that $\hat{W}(0,0,0)$ becomes 1. This is illustrated in Table 1. Suppose the operation $\hat{W}(0,0,0)$ is examined.

Table 1(a). Truth Table for Figure 2.

	x	
$z\ z'$	0	1
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1

Table 1(b). Truth Table for a Faulty Cell in Figure 2.

	x	
$z\ z'$	0	1
0 0	1	0
0 1	0	1
1 0	0	1
1 1	1	1

Case 1: Cell #1 is faulty.

Consider a test pattern of x=0, (z_1, z_1')=(0,0), and (z_j, z_j')=(0,1), j=2,....,n. For a fault-free ILA, a logical 0 is expected at the primary output W. When the pattern is applied to the faulty Cell #1, its output is changed from 0 to 1. This incorrect output will be propagated from Cell #2 to the rightmost cell. As a result, a logical 1 will appear at the primary output and conflict with the correct output 0. This concludes that, if the above test pattern is applied to the ILA under SCFM, a logic 1 at the primary output detects that one cell is faulty.

Case 2: The Cell #i, $i \neq 1$, is faulty.

Consider a test pattern of x=0, (z_i, z_i')=(0,0), and (z_j, z_j')=(0,1), for $1 \leq j \leq n$ and $j \neq i$. Similar to Case 1, a logical 0 is expected at the primary output of a fault-free ILA. When the above pattern is applied to the array, a correct output 0 will be propagated from the fault-free Cell #1 to #$i-1$ and an incorrect output produced by the faulty Cell #i will be propagated to the primary output. This concludes that, if the test pattern is applied to the ILA under SCFM, a logical 1 at the primary output indicates that one cell is faulty.

From above arguments, it is obvious that the examination of the operation $\hat{W}(0,0,0)$ requires n test patterns. Since it is necessary to examine all the possible operations of each cell, the number of test patterns required is thus proportional to n

(the length of the ILA). This concludes that the ILA of Figure 2 with cell function described by Table 1(a) is L-testable.

2.1.2. C-testable ILAs.

The concept of C-testability was first defined by Friedman [4]. He provided the following necessary and sufficient conditions for the C-testability of a 1-D ILA under SCFM:

Condition 1: In each test, all input combinations can be applied to every qth cell in the array by a single test.

Condition 2: The input sequence should be able to propagate the effect of the fault to an observable output.

According to *Condition* 1, the input patterns for each cell of an ILA must occur in a periodic manner. According to *Condition* 2, the function of the basic cell must be able to propagate the fault effect from its input to its output. These conditions can be studied more specifically with the following example. Consider the ILA of Figure 2 with cell function described in Table 2, which is the summation of a full adder.

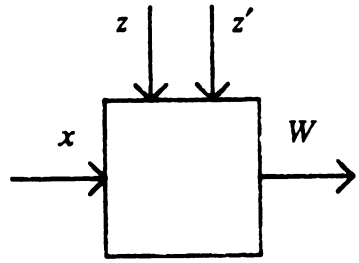
Table 2. Truth Table for Cells in a C-testable ILA.

	x	
$z\ z'$	0	1
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1

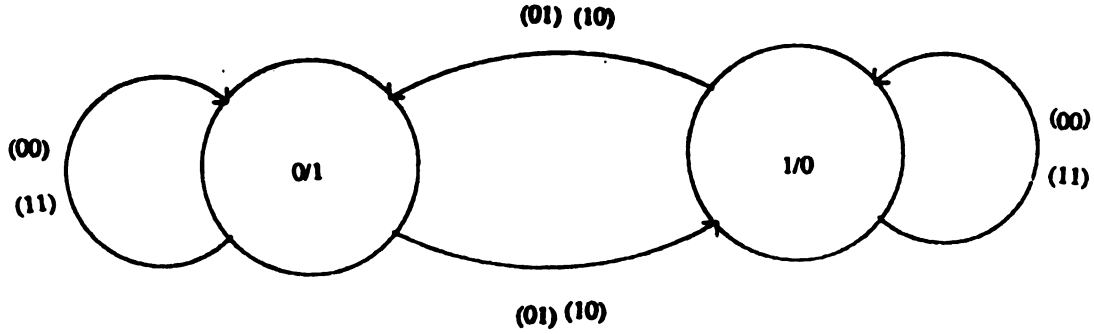
Figure 3(b) shows a fault pair diagram generated from a cell of Figure 3(a) with a function in Table 2. The state j/k represents a fault that changes the correct data j into k, where j and k are either 1 or 0. The parentheses indicates the input pair (z,z'). If the current state is 1/0, and if the input pair $(z,z')=(0,1)$ is applied, from Table 2, the W-output becomes 1 (0) when x-input is 0 (1). This concludes that the next state is 0/1. On the other hand, if the current state is 1/0 and if the input pair $(z,z')=(0,0)$, from Table 2, the W-output becomes 0 (1) when x-input is 0 (1), i.e., the next state is 1/0, or the state is not changed.

It is obvious that the fault pair diagram of Figure 3(b) is strongly connected. By [4], the ILA is C-testable. Alternatively, the C-testability can be also examined as follow.

If a test pattern of $x=1$ and $(z_i, z_i')=(0,1)$ for all i, as shown in Figure 4(a), is applied to an ILA with cells of Figure 3(a), then the outputs of the odd and even numbered cells are 0 and 1, respectively. Any single fault will generate an incorrect result



(a)



(b)

Figure 3. (a) A Cell with Fuction of Table 2;
(b) The Fault Pair Diagram of (a).

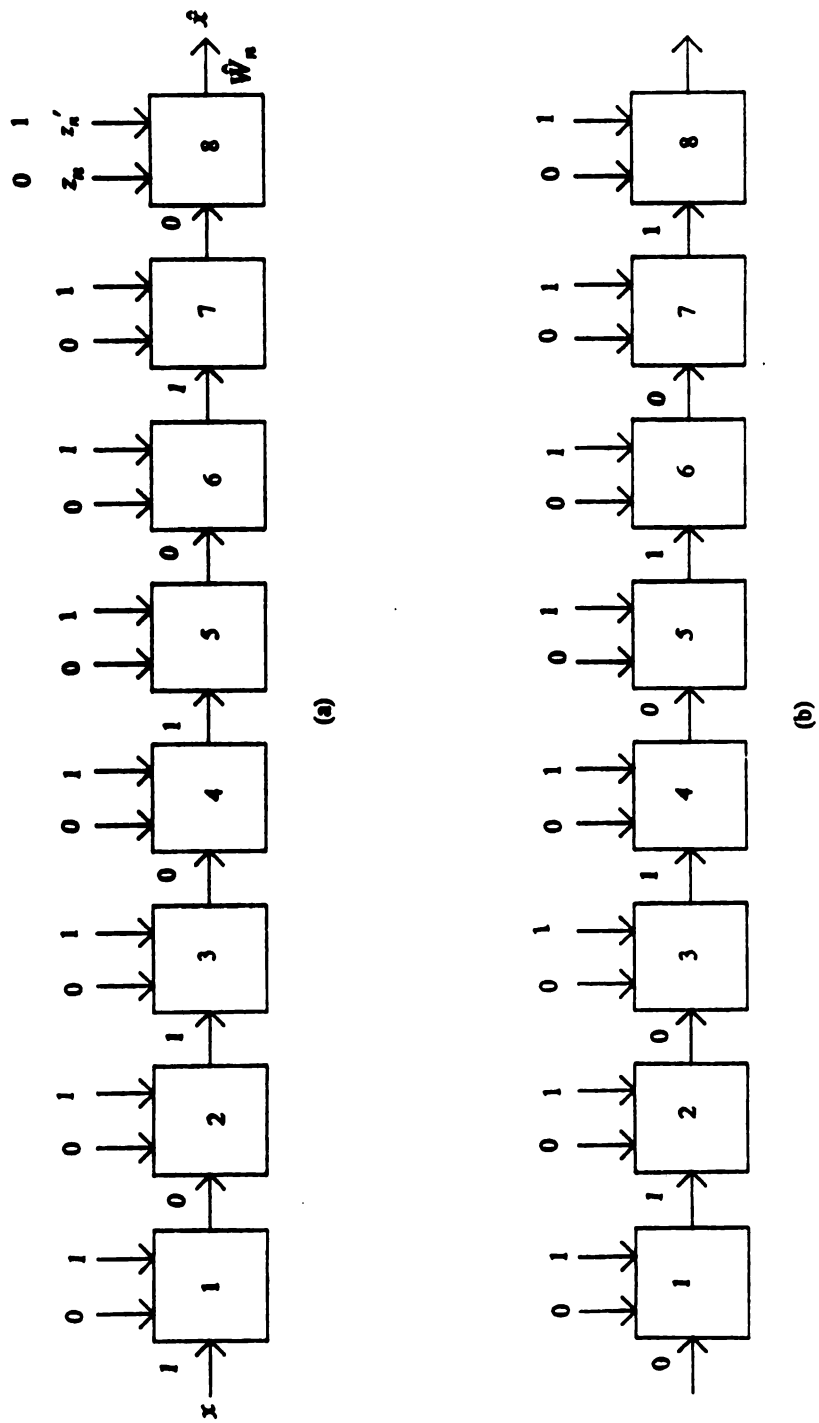


Figure 4. The testing of a ILA with input patterns (101) and (001).

at the primary output. For example, if a stuck-at-0 fault occurs at the output of Cell #2, then an incorrect output 0 is produced at the output of Cell #2 and further propagated to the primary output. This incorrect output conflicts with the expected output. In other words, the test pattern can simultaneously test any single stuck-at-0 (or stuck-at-1) fault at the output of any even (odd) numbered cells. In fact, if an additional test pattern of x=0 and (z_i, z_i')=(0,1), as shown in Figure 4(b), is also applied, then we can completely test both operations $\hat{W}(0,0,1)$ and $\hat{W}(1,0,1)$ in each cell of the ILA. This concludes that only 2 test patterns are required to test the ILA for these two operations, irrespective of the circuit size. If all the input combinations can be applied in this way, the ILA is thus C-testable.

As mentioned before, the ILA of Figure 2 with the cell function described by Table 1 is not C-testable. This can be studied by its fault pair diagram in Figure 5. Since both states 0/1 and 1/0 can be changed to the state 0/0 for input (z,z')=(0,0), or to the state 1/1 for input (z,z')=(1,1), which are not distinguishable faults. As a result, the ILA is not C-testable.

The problem arises as to whether or not the L-testable ILA of Figure 2 can be modified to be C-testable. Consider the ILA of Figure 6 which is modified from Figure 2. Each cell produces an additional output $\hat{S}_i$, where the output $\hat{S}_i$ is described in Table 2. In fact, both functions described in Tables 1 and 2 are respectively the carry and sum outputs of a *full adder* (FA). Figure 6 is known as a *ripple–carry adder*

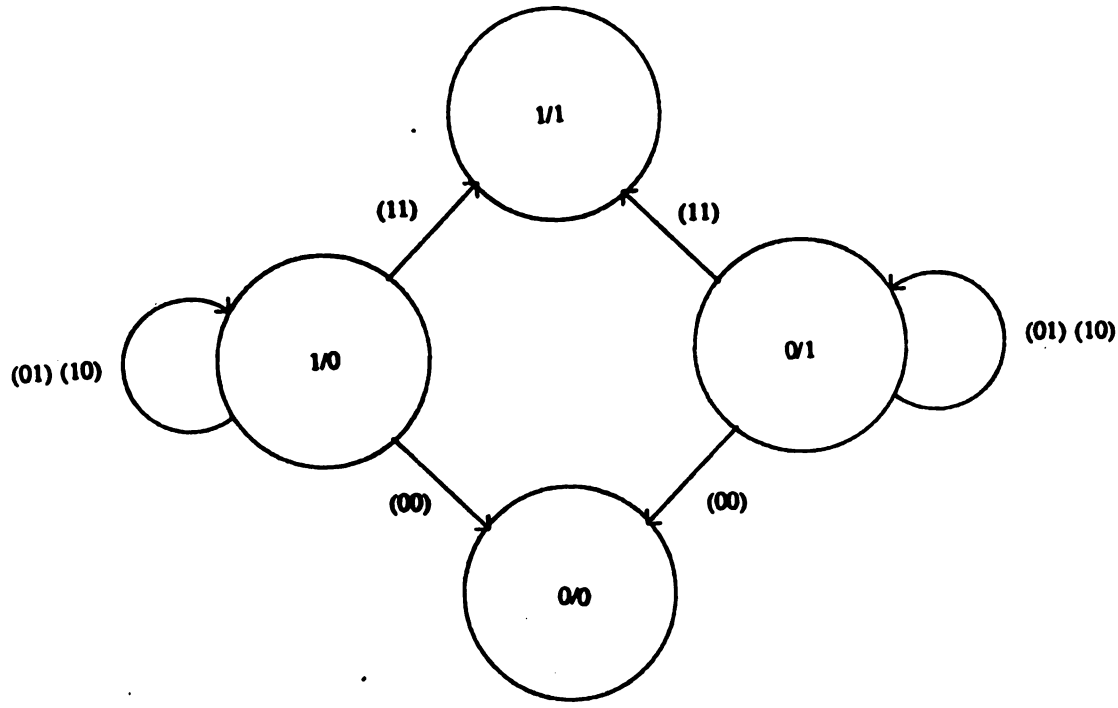


Figure 5. A Fault Pair Diagram of Table 1.

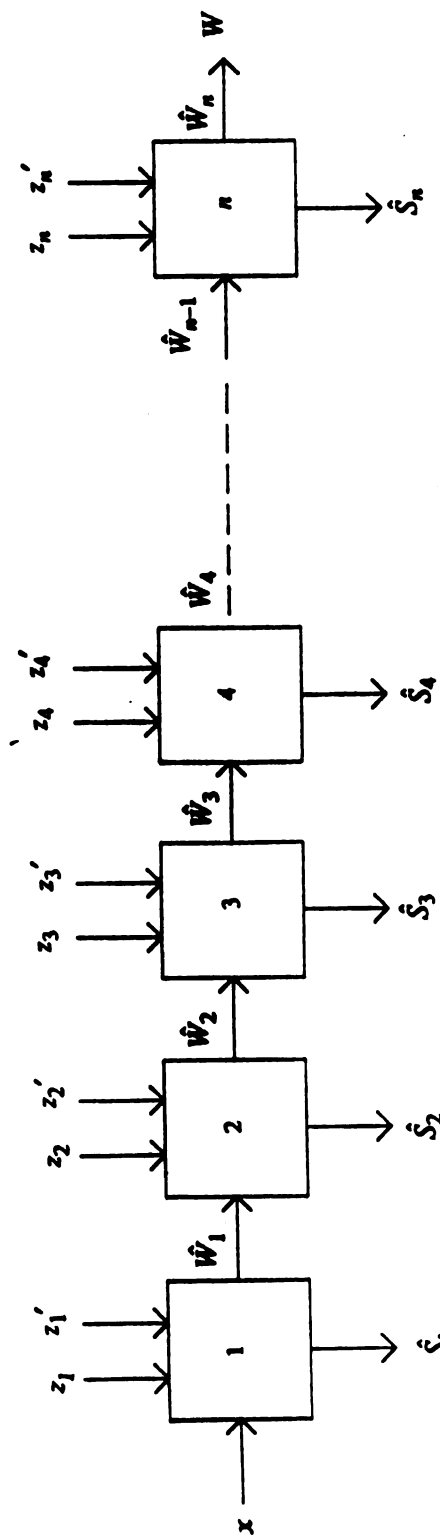


Figure 6. A 1-D ILA with primary outputs $\hat{S}_i$ and W.

(RCA). Since all the faults can be propagated out and observed from the additional primary outputs $\hat{S}_i$, the ILA of Figure 6 is thus C-testable. Therefore, Ripple carry adder is C-testable [9].

2.1.3. 2-D ILAs.

For 2-D ILA testing, various schemes have been recently proposed [10,11]. Basically, the 2-D ILA is partitioned into several 1-D rows and each row is treated as a 1-D ILA. Therefore, the C-testability developed for 1-D ILAs can be applied to 2-D ILAs.

Array multipliers and dividers are two special and simple forms of 2-D ILAs. It is known that an array multiplier is simply constructed by matrix of full adders with corresponding AND gates, the design methodology for C-testability of an array multiplier would be similar to that of a RCA, i.e., applying all the possible input combinations to every cell. Unfortunately, some array multipliers may not allow for the application of the complete set of input patterns to its basic cells. The conventional *Carry–Propagate Array Multipliers* (CPM), *Carry–Save Array Multipliers* (CSM), and *Baugh–Wooley Array Multipliers* (BWM) are, therefore, not C-testable [8].

Similarly, because array dividers do not allow for the application of the complete set of test patterns, neither the restoring array divider, nor the non-restoring array divider, is C-testable [9].

2.2. Previous Work.

Recently, the concept of C-testability developed for ILAs has been applied to the design of C-testable array multipliers [8]. Shen and Ferguson have shown that the testing of an array multiplier must involve the exhaustive testing of every cell by applying all possible input patterns and observing the outputs. In other words, for each cell consisting of an AND gate and a full adder, all possible 2^4 input patterns must be applied.

The significance of the design methodology of [8] is that they took advantage of the iterative structure in an array multiplier; the test sequence generated for exhaustively testing a cell can be applied to exhaustively test entire array. Consequently, the test length can be substantially reduced, and all cells can be simultaneously tested because of the repetitive nature of hardware. However, the only drawback in [8] is that no systematic test pattern generation procedure was provided.

In order to systematically generate the test sequence for C-testable iterative array structures, Chatterjee and Abraham [9] have proposed a test generation methodology using graph labeling scheme. A data-flow graph based model is formulated in which labels representing binary vectors are assigned to the branches of the data-flow graph. The labels are shown to satisfy a set of constraints imposed by cell function and interconnection topology. As a result, complex test generation problems can be solved by manipulating a set of symbolic labels with ease and efficiency [9].

2.2.1. Graph Labeling

Consider two sets of labels illustrated in Table 3 [9], which represent different sequence of 1's and 0's.

Table 3. Labels defined in [9].

V_1	V_2	V_3	V_4	C_1	C_2	C_3	C_4
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1

The combinations of the vectors V_1, V_2, and V_3 contain all 2^3 possible binary 3-bit values. V_4 is the bitwise sum of V_1, V_2, and V_3 over GF(2). Two functions are defined as

$$g(V_i,V_j,V_k)=V_i \oplus V_j \oplus V_k \; ; \tag{1}$$

and

$$f(V_i,V_j,V_k)=V_iV_j+V_iV_k+V_jV_k \tag{2}$$

The function g is the bitwise summation over GF(2) of the vectors V_i, V_j, and V_k, while the function f is the bitwise carry produced in the above summation. The C_m vectors are computed by evaluating $f(V_i,V_j,V_k)$ and the V_m vectors are by $g(V_i,V_j,V_k)$, where $i \neq j \neq k \neq m$, and $1 \leq i,j,k,m \leq 4$.

Let A, B, C, X, and Y be vectors that represent any of 8 vectors V_1-V_4, and C_1-C_4. A mapping VM is defined as VM(A,B,C)=XY if X=g(A,B,C) and Y=f(A,B,C). It is represented simply by ABC-->YX.

The vectors V_i's and C_j's are treated as labels. A data flow graph representation of the circuit is used. Each node of this graph represents a circuit module and interconnection between the modules is represented by directed arcs between the above nodes. The objective is to keep track of the data in each branch by the following way :

1) each branch is assigned an unique label; and

2) the labels on the input and output branches of a node are consistent with the corresponding cell function.

2.2.2. Design of a C-testable CPM.

Consider the 4-by-4 Carry-Propagate array multiplier (CPM) [9], as shown in Figure 7. Each cell has 3 inputs, x, y, and z, and 2 outputs, sum bit $u=x\oplus y\oplus z$ and carry bit $v=xy+xz+yz$. The input x is the AND a_ib_j, where a_i's and b_j's are the multiplier and multiplicand bits, respectively.

In [9], a label L consisting of two sets of vectors $L_1=(V_1,C_1,V_4)$ and $L_2=(V_4,C_4,V_1)$ is applied to a CPM, as shown in Figure 8(a). The mapping,

$$V_1C_1V_4\text{-->}V_4C_4\ ,$$

describes that a carry vector V_4 is propagated from the rightmost cell to the leftmost cell of the first row. In order to reproduce the label L_1 in the third row, an appropriate label L_2 is chosen for the second row such that the carry assigned in the rightmost cell of the second row can be propagated to the leftmost cell and the labels L_1 and L_2 are periodically reproduced in every other row. This is referred to as *two–row periodic propagation* (TRPP).

Since the labels on the input and output branches of a graph node must be consistent with the corresponding cell function, the carry vector of the leftmost cell in the first row must be identical to the required vector in the y input of the leftmost cell in the second row. On the other hand, the sum output of each cell in the first row is

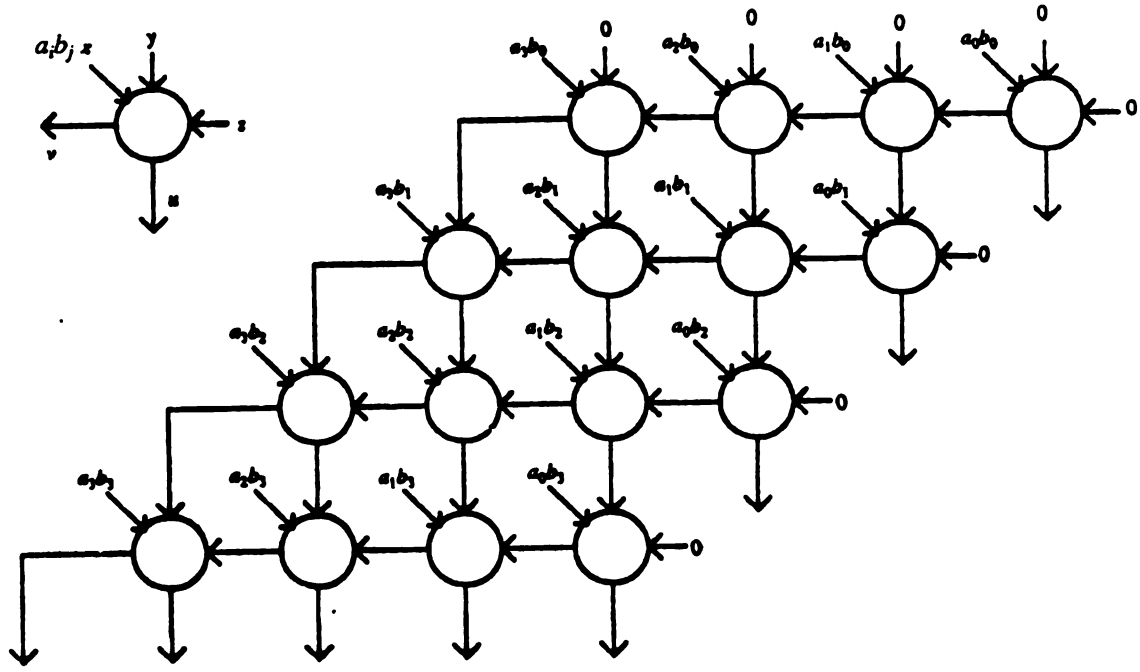


Figure 7. A Schematic Circuit Diagram of 4-by-4 Carry-Propagate Array Multiplier [9].

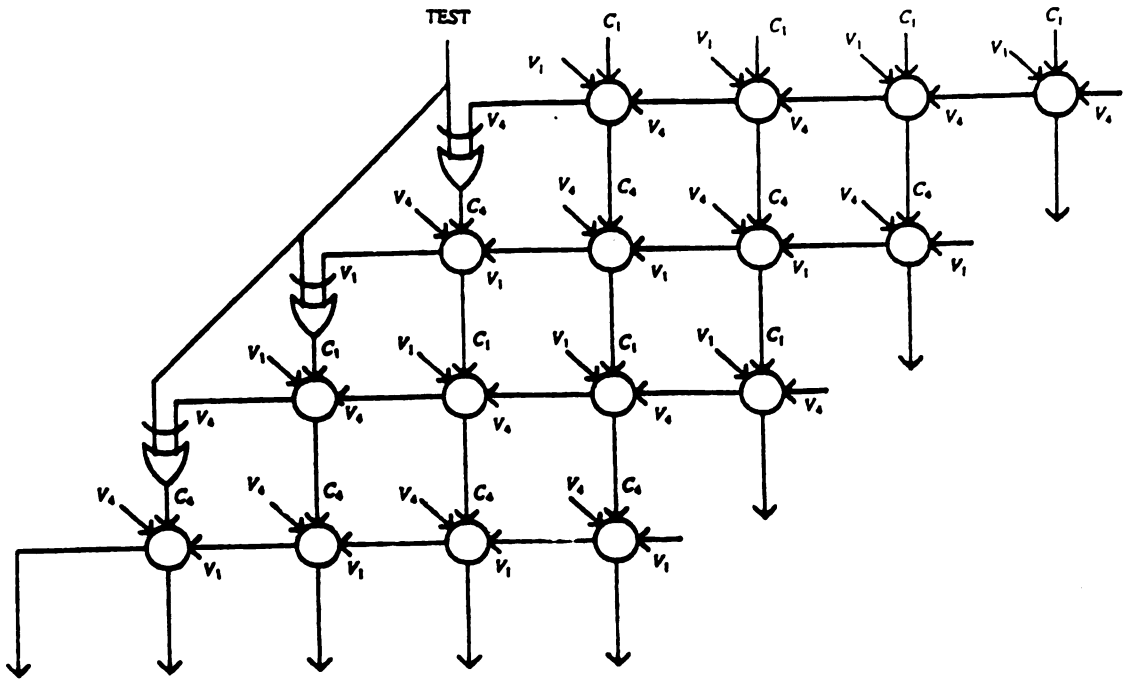


Figure 8(a). Labeling L for Carry-Propagate Array Multiplier [9].

Table 4(a). Application of L to CPM [9]

	Vector 1				Vector 2				
Valid	V_1	C_1	V_4	a_ib_j	V_4	C_4	V_1	a_ib_j	TEST
•	0	0	0	00,10	0	0	0	00,10	0
-	0	1	1	10	1	0	0	11	1
•-	0	1	1	10	1	0	0	11	1
•	0	1	0	00,10	0	1	0	00,10	1
•	1	0	1	11	1	0	1	11	1
-	1	0	0	11	0	1	1	10	1
•	1	0	0	11	0	1	1	10	1
•	1	1	1	11	1	1	1	11	0

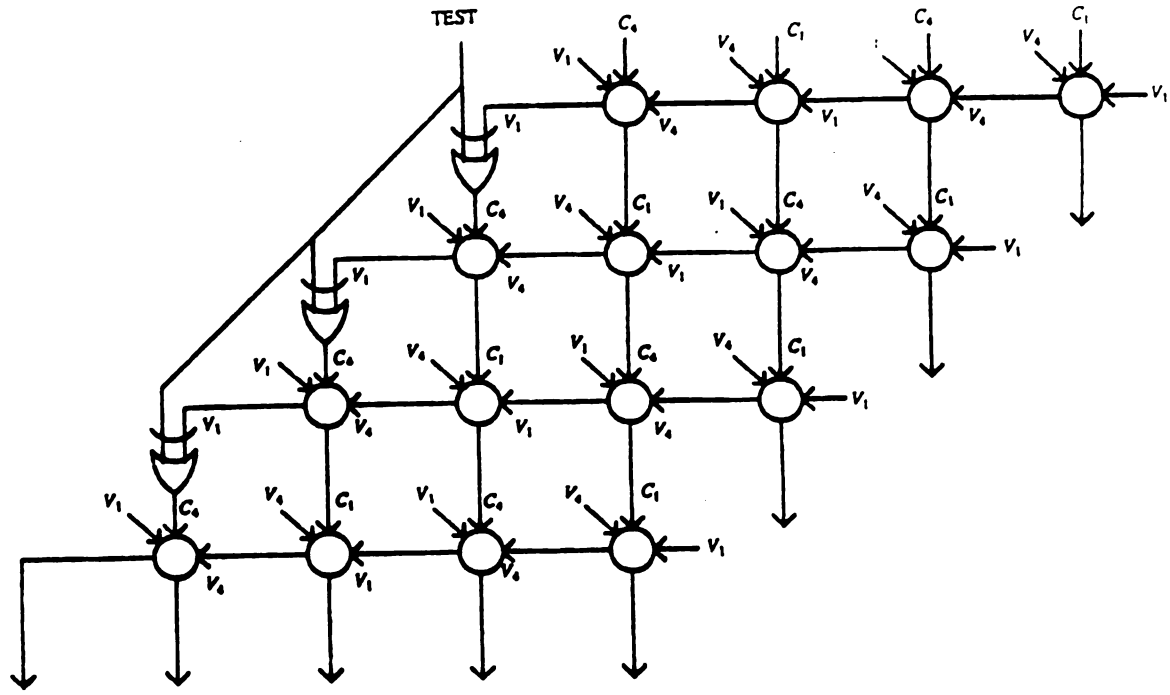


Figure 8(b). Labeling L' for Carry-Propagate Array Multiplier [9].

Table 4(b). Application of L' to CPM [9]

	Vector 1				Vector 2				
Valid	V_1	C_4	V_4	a_ib_j	V_4	C_1	V_1	a_ib_j	TEST
-	0	0	0	01	0	0	0	01	0
*	0	0	1	01	1	1	0	11	0
-	0	0	1	01	1	1	0	11	0
-	0	1	0	01	0	1	0	01	0
-	1	0	1	11	1	0	1	11	0
*	1	1	0	11	0	0	1	01	0
-	1	1	0	11	0	0	1	01	0
-	1	1	1	11	1	1	1	11	0

fed to the y input of the corresponding cells in the second row. Therefore, the vectors must be chosen so that both carry and sum outputs have the same label. Unfortunately, producing such labels is virtually impossible for this application.

However, with an additional XOR gate, the labels can be perfectly applied as shown in Figure 8. More specifically, the leftmost cell in the first row produces the carry vector V_4 and the sum vector C_4. The carry vector is expected to be fed into the y input of the leftmost cell in the second row where an input vector C_4 is expected. Although vectors V_4 and C_4 are not identical, C_4 is bitwise complement of V_4 except when (V_1,V_2,V_3)=(000) and (111). Therefore, an additional two-input XOR gate can be used to make the applied label consistent, as shown in Figure 8, where an extra control signal TEST is needed. The signal TEST is set to a logical 0 during the normal operation and is set to 1 when the corresponding values of V_4 and C_4 are different during the test mode.

The use of label L, however, is not enough to apply all possible input combinations. As the application of L to array illustrated in Table 4(a), all input combinations are applied except (001) and (110) in Vector 1. Therefore, another label L' consisting of $L_1'=(V_1,C_4,V_4)$ and $L_2'=(V_4,C_1,V_1)$ is applied. This label is applied in the same manner as L except that the labels L_1' and L_2' are periodically reproduced in every other diagonal column, as shown in Figure 8(b) [9]. This is referred to as *two–column periodic propagation* (TCPP).

With the application of such labels, the modified Carry-propagate array multiplier has been shown to be C-testable [9].

2.3. Problem Description.

In this thesis, the following three problems will be discussed.

(1) Designs of C-testable Array Multipliers.

Although the C-testable CPM design of [9] can significantly reduce the time and cost of testing, it is not free of penalty. The extra XOR gates may slightly degrade the speed performance. Alleviating the performance degradation is desirable. In addition, the graph labeling scheme for both CSM and BWM were not discussed in [9]. However, these array multipliers are the most commonly used. Therefore, the design of C-testable CSM and BWM is proposed.

(2) Designs of C-testable array dividers.

An array divider design has been presented and claimed to be C-testable in [9]. However, due to the difficulty of fault propagation, the design of the non-restoring array divider proposed in [9] is, in fact, not C-testable. The design of C-testable array divider is studied.

(3) Design Methodologies.

Since the methodology of generating test patterns and expected outputs has not been precisely stated and provided in the existing literature, the algorithms that generate test patterns and expected outputs for arithmetic units are thus investigated.

III. Design and Test of C-testable Array Multipliers

3.1. Carry-Propagate Array Multipliers (CPM).

According to graph labeling scheme, the following properties have been summarized [9].

Property 1: Any combination of three vectors $V_iC_jV_k$ or $C_iV_jC_k$, $1 \le i,j,k \le 4$, $i \ne j \ne k$, does not contain the 3-bit combinations 010 and 101.

Property 2: The set of vectors $V_iC_iV_j$ ($C_iV_iC_j$) and its dual $V_jC_iV_i$ ($C_jV_iC_i$) together contain all possible combinations of 3-bit values.

Property 3: The set of vectors $V_iV_jV_k$ and $C_iC_jC_k$ where $1 \le i,j,k \le 4$ and $i \ne j \ne k$, cover all the possible 3-bit combinations.

Our goal is to find a set of labels that can completely apply all possible input combinations to each cell. While the labels in *Property* 1 cannot apply all input combinations, the labels in *Property* 2 will result a performance degradation. Therefore, we shall consider the labels that are comprised of either all vectors of $\{V_1,V_2,V_3,V_4\}$, or all vectors $\{C_1,C_2,C_3,C_4\}$.

3.1.1. Graph Labeling.

Consider the corresponding mappings,

$$V_iV_jV_k \longrightarrow C_mV_m \text{ ; and} \tag{3}$$
$$C_iC_jC_k \longrightarrow V_mC_m \text{ ,} \tag{4}$$

where $i \neq j \neq k \neq m$, and $1 \leq i,j,k,m \leq 4$, which represent the functions of the basic cell in a CPM. Consider also the application of the labels that are propagated in the fashion of combining TRPP and TCPP, i.e., two vectors are periodically propagated in one direction and the other two are in the other direction. More specifically, let M_i's, i=1,..,4, represent the four basic cells of a CPM, as shown in Figure 9. Each cell is labeled by $L_i=(L_{i1},L_{i2},L_{i3})$. The objective is to generate a set of labels so that they can be propagated to the entire array repetitively. According to their interconnection topology and the mappings (3) and (4), we shall solve the following mappings,

$$L_{11}L_{12}L_{13} \longrightarrow L_{23}L_{42} \text{ ;} \tag{5}$$
$$L_{21}L_{22}L_{23} \longrightarrow L_{13}L_{32} \text{ ;} \tag{6}$$
$$L_{31}L_{32}L_{33} \longrightarrow L_{43}L_{22} \text{ ; and} \tag{7}$$
$$L_{41}L_{42}L_{43} \longrightarrow L_{33}L_{12} \text{ .} \tag{8}$$

This results the following theorem.

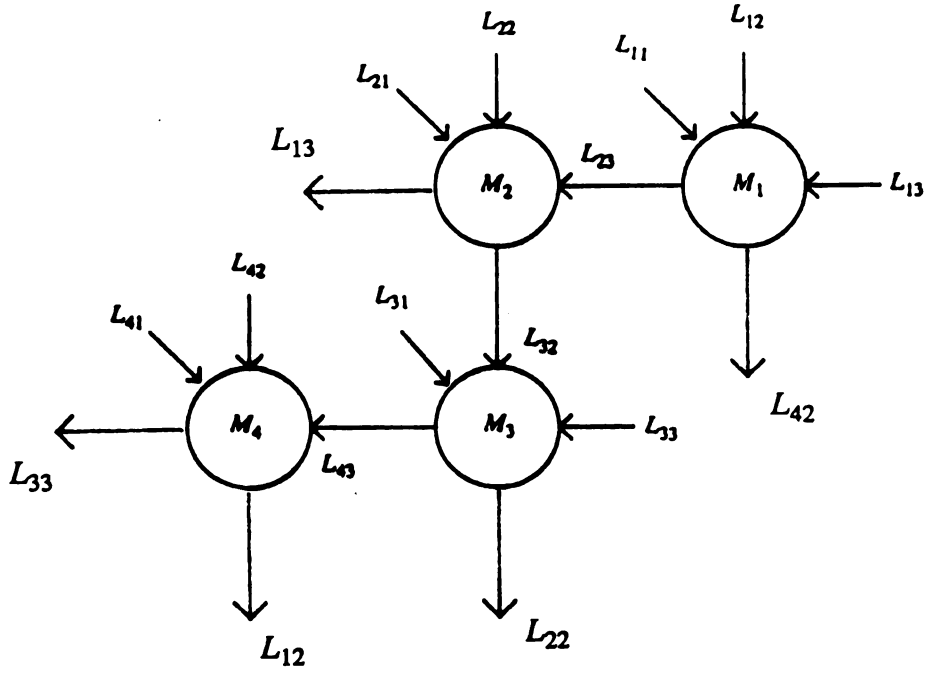


Figure 9. Four Basic Cells of a CPM.

Theorem 1.

The following set of labels is a solution of the mapping (5)-(8) :

$$
\begin{aligned}
&L_1=(L_{11},L_{12},L_{13})=(V_1,V_2,V_3);\\
&L_2=(L_{21},L_{22},L_{23})=(C_2,C_1,C_4);\\
&L_3=(L_{31},L_{32},L_{33})=(C_4,C_3,C_2);\\
&L_4=(L_{41},L_{42},L_{43})=(V_3,V_4,V_1).
\end{aligned}
\tag{9}
$$

Lemma 1.

Consider four distinct indices i, j, k, and m, $1\le i,j,k,m\le 4$, for labels V and C, and the mapping: xyz --> vu, where x, y, and z belong to either all V's or all C's, i.e., $(x,y,z)=(V_i,V_j,V_k)$ or (C_i,C_j,C_k), then we get the following properties.

(a) The indices of the vectors for u and v are the same, i.e.,

iff $u=V_m$, then $v=C_m$, and iff $u=C_m$, then $v=V_m$.

(b) If any one of x, y, and z is C_k and $u=C_m$ or $v=V_m$,

then the others belong to $\{C_i,C_j\}$, and

(b') If any one of x, y, and z is V_k and $u=V_m$ or $v=C_m$,

then the others belong to $\{V_i,V_j\}$.

(c) If one of x, y, and z is V_i, then u$\neq V_i$, and v$\neq C_i$, and

(c') If one of x, y, and z is C_i, then u$\neq C_i$, and v$\neq V_i$.

(d) If v=C_m or u=V_m, then none of x, y, and z is V_m, and

(d') If v=V_m or u=C_m, then none of x, y, and z is in C_m.

Proof: The above results can be simply obtained from the mappings $V_iV_jV_k \longrightarrow C_mV_m$ and $C_iC_jC_k \longrightarrow V_mC_m$ in (3) and (4).

Proof of Theorem 1 :

If $L_1=(V_1,V_2,V_3)$, by equations (3) and (5), we get $L_{23}=C_4$ and $L_{42}=V_4$. Further, by equation (8) and Lemma 1(a), $L_{12}=V_2$ results in $L_{33}=C_2$. Similarly, since $L_{13}=V_3$, by equation (6) and Lemma 1(a), we conclude that $L_{32}=C_3$. On the other hand, since $L_{23}=C_4$ and $L_{13}=V_3$, by equation (6) and Lemma 1(b), L_{21} and L_{22} are identical to C_1 and C_2. Similarly, since $L_{12}=V_2$ and $L_{42}=V_4$, by equation (8) and Lemma 1(b'), L_{41} and L_{43} are identical to V_1 and V_3. Moreover, by equation (7) and Lemma 1(c'), $L_{33}=C_2$ gives $L_{22}\neq C_2$ and forces $L_{22}=C_1$ and $L_{21}=C_2$. Again equation (7) and Lemma 1(a), $L_{22}=C_1$ implies $L_{43}=V_1$ and further forces $L_{41}=V_3$. Consequently, from the results, $L_{22}=C_1$, $L_{32}=C_3$, and $L_{33}=C_2$, we conclude that $L_{31}=C_4$ by equation (7) and Lemma 1(b').

Corollary 1.1.

Given a label L_1 consisting of either all V_i, or all C_i, the rest of labels can be generated in the same fashion as discussed in Theorem 1.

Proof: Consider an index set {1,2,3,4}. Given a label $L_1=(V_i,V_j,V_k)$. If we permute the indices i,j,k, and m, i.e., assign i to 1, j to 2, k to 3, and m to 4, then we get the labels

$$L_1=(V_i,V_j,V_k),\ L_2=(C_j,C_i,C_m),\ L_3=(C_m,C_k,C_j),\ \text{and}\ L_4=(V_i,V_m,V_k). \tag{10}$$

Similarly, if $L_1=(C_i,C_j,C_k)$ then

$$L_2=(V_j,V_i,V_m),\ L_3=(V_m,V_k,V_j),\ \text{and}\ L_4=(C_i,C_m,C_k). \tag{11}$$

Corollary 1.2.

There exist 24 possible sets of such labels.

Proof: Since label L_1 takes three indices out from a set of four, this implies that there exist 12 possible sets. Furthermore, since L_1 can take either all V_i, or all C_i, this results in a total of 24 sets.

In fact, each set of labels in *Corollary* 1.2 contains the same 8 combinations for the 3-bit input, but in a different sequence.

Figures 10(a) and 10(b) respectively illustrate the applications of label (9) to a 4-by-4 and a 5-by-5 CPM. The labels are perfectly applied and periodically propagated through the cells of the entire array except those in the carry propagation stage, i.e., the leftmost cells of each row, referred to as *left–boundary cells*. Although the labels applied to the left-boundary cells are not exactly the same as expected, they have the same elements in the label but in different sequence. More specifically, the label of the leftmost cell in the second row of Figure 10(a) was expected to be (x,y,z)=(V_3,V_4,V_1) and now is changed to (V_4,V_3,V_1), referred to as *Vector* 6. Similarly, in the leftmost cell of the third row, the label (C_2,C_2,C_4) is changed to (C_1,C_2,C_4), referred to as *Vector* 5. Table 5 describes the input combinations of these four vectors. Each pair contains the all possible 8 input combinations in different sequence.

Table 5. Input Combinations of Boundary Cells.

	Vector 4 (V_3,V_4,V_1)	Vector 6 (V_4,V_3,V_1)	Vector 2 (C_2,C_1,C_4)	Vector 5 (C_1,C_2,C_4)
1	0 0 0	0 0 0	0 0 0	0 0 0
2	1 1 0	1 1 0	1 1 0	1 1 0
3	0 1 0	1 0 0	0 1 0	1 0 0
4	1 0 0	0 1 0	0 1 1	1 0 1
5	0 1 1	1 0 1	1 0 0	0 1 0
6	1 0 1	0 1 1	1 0 1	0 1 1
7	0 0 1	0 0 1	0 0 1	0 0 1
8	1 1 1	1 1 1	1 1 1	1 1 1

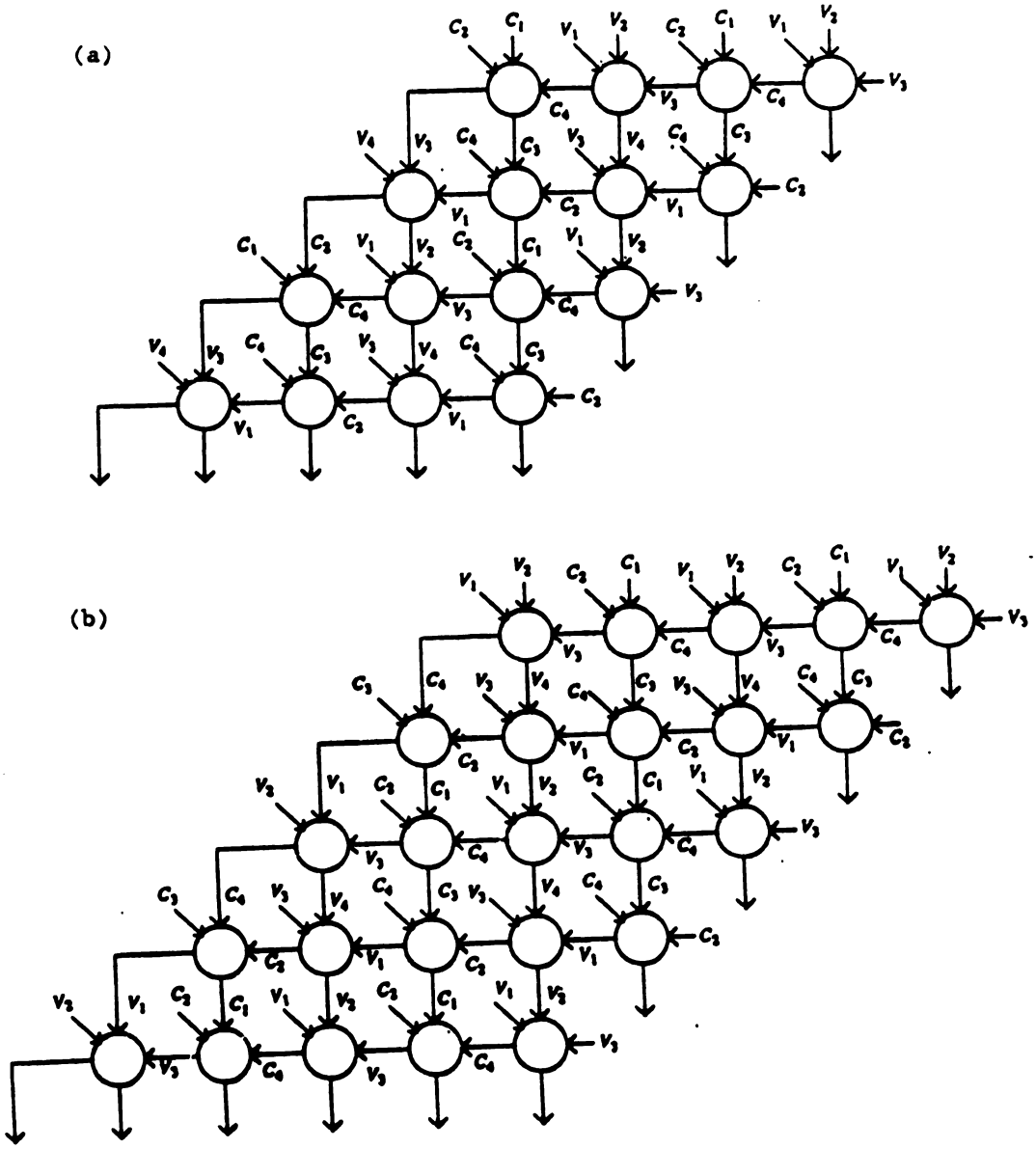


Figure 10. Labeling for (a) a 4-by-4 CPM; (b) a 5-by-5 CPM;
(c) A Schematic Diagram of a 4-by-4 modified CPM.

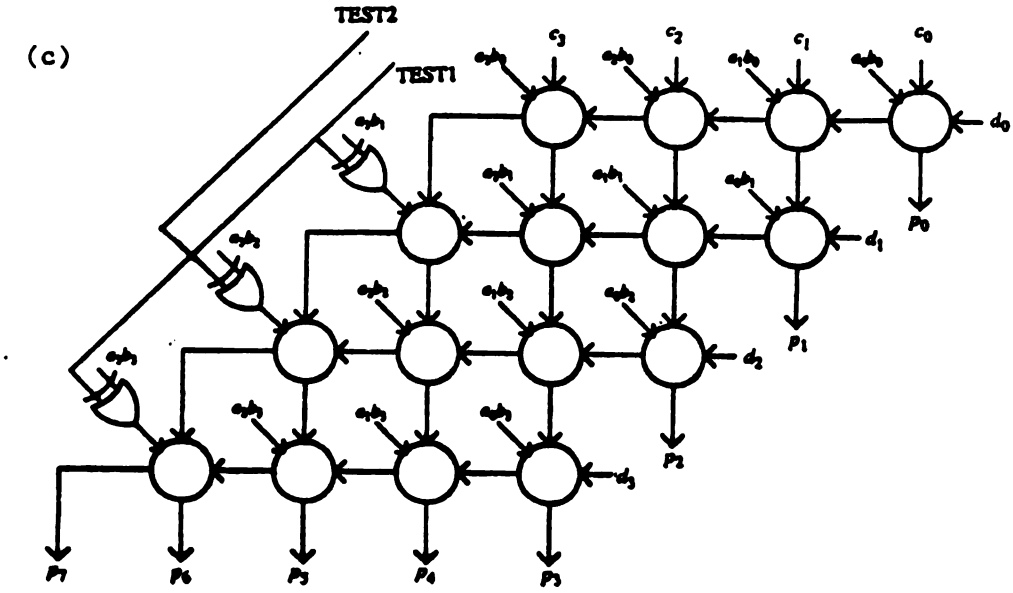


Figure 10. (Continued)

The problem arises as to whether or not the application of such labels meets the constraints, referred to as *external constraints*, imposed by cell functions and interconnection topology.

As the interconnection topology shown in Figure 7, the x-direction input of each cell is the output of a 2-input AND gate. The inputs, a_i and b_j of the AND gate, must meet the following constraints: all cells in the same diagonal column are required to apply the same a_i, and all cells in the same row have the same b_j. Therefore, both vectors 1 and 3 have the same value of a_i, so do vectors 2 and 4; and both vectors 1 and 2 have the same value of b_j, so do vectors 3 and 4. More specifically, (a_i,b_j)=(1,1) if the output of AND gate is a logical 1; otherwise (a_i,b_j)=(0,0), (0,1), or (1,0) depending upon the external constraints and the corresponding data of L_{i1}. Table 6 describes the application of the labels of equation (9) to the array and get the suitable a_i's and b_j's.

Table 6. Application of Labels L_i's to the array.

Vector 1					Vector 2					Vector 3					Vector 4				
a_i	b_j	V_1	V_2	V_3	a_i	b_j	C_2	C_1	C_4	a_i	b_j	C_4	C_3	C_2	a_i	b_j	V_3	V_4	V_1
*		0	0	0	*		0	0	0	*		0	0	0	*		0	0	0
0	1	0	0	1	1	1	1	1	0	0	1	0	0	1	1	1	1	1	0
*		0	1	0	*		0	1	0	*		0	1	0	*		0	1	0
1	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	0	0
1	1	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1
1	1	1	1	0	0	1	0	0	1	1	1	1	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Remark: "*" denotes that a_ib_j can be either 00, 01,or 10.

From Table 6 and Table 5, we may find that both a_i and b_j can be applied consistently to meet the external constraints for all cells in the array except those left-boundary cells. Since Vector 4 is substituted by Vector 6 and Vector 2 by Vector 5 for those boundary cells, applying the suitable a_i and b_j of Vector 4 to the corresponding left-boundary cells will produce a Vector (V_3,V_3,V_1) which is not Vector 6. Similarly, applying the suitable a_i and b_j of Vector 2 to the corresponding left-boundary cells will not produce the Vector 5. Therefore, the problem can be solved by adding an XOR gate as shown in Figure 10(c) with a control signal.

In order to exhaustively test all the cells of a CPM, all the possible inputs should be examined. Table 7 describes the input combinations derived from Table 6.

Table 7. Input Combinations for MCPM.

Test_#	Vector 1 (a b x y)	Vector 2 (a b x y)	Vector 3 (a b x y)	Vector 4 (a b x y)
1.	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2.	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
3.	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0
4.	0 1 0 1	1 1 1 0	0 1 0 1	1 1 1 0
5.	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0
6.	0 1 1 0	0 1 1 0	0 1 1 0	0 1 1 0
7.	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0
8.	1 0 1 1	1 0 1 1	1 1 0 0	1 1 0 0
9.	1 1 0 0	1 1 0 0	1 0 1 1	1 0 1 1
10.	1 1 0 1	1 1 0 1	1 1 0 1	1 1 0 1
11.	1 1 1 0	0 1 0 1	1 1 1 0	0 1 0 1
12.	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

Table 7 shows that all input combinations are applied to the basic cell except (a_i,b_j,y,z)=0001, 1001, 0011, and 0111. As discussed in [8] and Lemma 2, the patterns (0001) and (1001) can never appear at the inputs to a cell under normal operation.

Lemma 2. [8]

The input vectors (1001) and (0001) can never appear at the input to any cell in the CPM.

Proof: Suppose that the input pattern (1001) is applied to the cell at the ith diagonal column and the jth row, say, Cell(i,j). (1001) represents that a_i=1, b_j=0 and the z-input is 1. The z-input of Cell(i,j) is nothing but the carry output u of Cell(i−1,j). Since the external constraint, b_j=0, results in a zero at the x-input of Cell(i−1,j), both z and y-inputs must be 1 to produce the carry output u=1. Similarly, the z-input of Cell(k,j) must be 1 for all k, where $1 \le k \le i$. However, the z-input of Cell(1,j) is fed a logical 0 as shown in Figure 7. Therefore, the input pattern (1001) will never appear during normal operation. Similarly, the pattern (0001) will never appear during normal operation.

In order to apply all possible input combinations, the cell is modified in such a way that the carry output of the input (0001) is changed from logical 0 to 1 [8]. As a result, the use of the combination (0001) can apply the combinations (0011) and (0111) to the array. Therefore, the following input combinations are added to Table 7.

Table 7(a). Input Combinations for MCPM.

13.	0 0 0 1	0 0 1 1	0 0 0 1	0 0 1 1
14.	0 0 1 1	0 0 0 1	0 0 1 1	0 0 0 1
15.	0 0 0 1	0 0 0 1	0 1 1 1	0 1 1 1
16.	0 1 1 1	0 1 1 1	0 0 0 1	0 0 0 1

According to Theorem 1, a set of test vectors can be obtained in Table 7, i.e., assign column a of Vector i to be L_{i1}, column b of Vector i to be L_{i2}, and L_{i3} can be generated by column y and z of Vector i. The entries in column y and z are input data. If the tester can apply those labels on the accessible inputs, then the test vectors in Table 7 can be propagated to the entire array repetitively. Therefore, the following theorem results.

Theorem 2.

The MCPM is C-testable with a test length of 16.

Lemma 3.

A Basic FA/AND cell of CPM shown in Figure 7 can be tested with 16 patterns.

Proof: A 4-input circuit can be exhaustively tested by all its input combinations, i.e., 2^4=16 test patterns.

Lemma 4.

The four basic cells of CPM shown in Figure 9 can be tested with 16 patterns.

Proof: From the generation of Table 7, each basic cell can be exhaustively tested by those 16 patterns.

Proof of Theorem 2 :

The MCPM is C-testable if, for any size n, a n-by-n MCPM can be tested with 16 patterns. Let M_{BC} be the four basic cells of CPM, as shown in Figure 9, and TS be the 16 test vectors of Table 7. A p-by-q MCPM represents a MCPM having p rows and q columns. Without loss of generality, both p and q are assumed to be even. Therefore, a p-by-q MCPM can be tessellated by M_{BC}'s. Consider a 2-by-(2k) MCPM which is constructed by k M_{BC}'s, say MR_1, MR_2, ..., MR_k. From Figure 9, the inputs of MR_j, j=2...k, are the same as those of MR_1. Therefore, all MR_j can be tested by the same test set TS. Similarly, a (2r)-by-2 MCPM can be constructed by r M_{BC}'s, say MC_1, MC_2, ..., MC_r. From Figure 9, the inputs of MC_i, i=2...r, are the same as those of MC_1. Therefore, all MC_i can be tested by the same test set TS. Since, by Lemma 4, each M_{BC} can be tested by the test set TS, a n-by-n MCPM that can be partitioned into m^2 M_{BC}'s, where n=2m, can then be tested by the same test set TS. So, MCPM is C-testable with a test length of 16.

3.1.2. Test Pattern Generation.

Consider a 4-by-4 modified carry propagate array multiplier (MCPM), as shown in Figure 10(c). The c's and d's inputs and control signals are connected to logical 0 during multiplication. However, it is assumed that during testing these inputs are available as primary inputs to the array. Two control signals TEST1 and TEST2 are connected to the even and odd numbered rows, respectively.

According to the input combinations of Table 7, Algorithm 1 describes the process of generating the test patterns and the corresponding expected outputs for a C-testable MCPM. The main idea of Algorithm 1 is that the test sequence designed by Theorem 1 for testing the four basic CPM cells can be applied to the entire array. Therefore, the test vectors that exhaustively test the four basic cells of CPM can be propagated to all the other cells. Algorithm 1 simply apply those test vectors in Table 7 into an MCPM repetitively.

Algorithm 1:

```
{*  Vector_i=(ia,ib,ic,id), i=1,..,4.
 *        ia : a-input, ib : b-input, ic : y-input, id : z-input.
 *  The Test patterns to be generated are:
 *    a(i), b(i), c(i), d(i),i =0...n, TEST1, TEST2.
 *  The expected product p(i), i=0...2n+1.
 *  n is odd.     *}

{*  Step 1. (Test Patterns)       *}

  For i=0 to n-1 by 2
      do Begin
        a(i):=1a; a(i+1):=2a;
        b(i):=1b; b(i+1):=3b;
        c(i):=1c; c(i+1):=2c;
        d(i):=1d; d(i+1):=3d;
      End;

{*  (consider the leftmost cell of the second row) *}

  If ((1d XOR 4d)=1c)  Then x_input:=0 Else x_input:=1;
  If (a(n)*b(1)=x_input)  Then TEST1:=0  Else TEST1:=1;

{*  (consider the leftmost cell of the third row) *}

  If ((4c XOR 2d)=3c)  Then x_input:=0  Else x_input:=1;
  If (a(n)*b(2)=x_input)  Then TEST2:=0  Else TEST2:=1;

{*  Step 2: (Expected Results)     *}

  For i=0 to n-1 By 2
      do Begin
        p(i):=4c; p(i+1):=2c;
      End;
  For i=n+1 To 2n+1 By 2
      do Begin
        p(i):=2c; p(i+1):=1c;
      End;
```

The test patterns and expected outputs for a 4-by-4 MCPM are generated as shown in Table 8.

Table 8. Test Patterns and Expected Outputs for a MCPM.

Test_#	a	b	c	d	TEST 1 2	Expected Output
1	0000	0000	0000	0000	0 0	00000000
2	0000	1111	0000	0000	0 0	00000000
3	1111	0000	0000	0000	0 0	00000000
4	1010	1111	1010	1111	0 0	10101111
5	0000	0000	1111	0000	1 1	01111111
6	0000	1111	1111	0000	1 1	01111111
7	1111	0000	1111	0000	1 1	01111111
8	1111	1010	1111	0101	1 1	01111010
9	1111	0101	0000	1010	1 1	10000101
10	1111	1111	0000	1111	1 1	10000000
11	0101	1111	0101	0000	0 0	01010000
12	1111	1111	1111	1111	0 0	11111111
13	0000	0000	1010	1111	0 0	10101111
14	0000	0000	0101	1111	1 1	11010000
15	0000	1010	0000	1111	0 1	10000101
16	0000	0101	1111	1111	1 0	11111010

Remarks : a=($a_3a_2a_1a_0$) b=($b_3b_2b_1b_0$) c=($c_3c_2c_1c_0$) d=($d_0d_1d_2d_3$)

By applying all the test patterns in Table 8, Table 9 illustrates that all the input combinations can be applied to each cell of the array. This shows that, the test patterns in Table 8 can detect any single fault that occurs at any place of the array.

Table 9. Input Combinations for each cell in a 4-by-4 CPM.

#1				0000	0000	0000	0000	#9				1100	1100	1100	1100
			000	0000	0000	0000					101	1011	1011	1011	
		000	0000	0000	0000					010	1100	1100	1100		
	000	0000	0000	0000					101	1011	1011	1011			
#2				0100	0100	0100	0100	#10				1101	1101	1101	1101
			000	0100	0100	0100					011	1101	1101	1101	
		000	0100	0100	0100					011	1101	1101	1101		
	000	0100	0100	0100					011	1101	1101	1101			
#3				1000	1000	1000	1000	#11				0101	1110	0101	1110
			000	1000	1000	1000					001	1110	0101	1110	
		000	1000	1000	1000					001	1110	0101	1110		
	000	1000	1000	1000					001	1110	0101	1110			
#4				1110	0101	1110	0101	#12				1111	1111	1111	1111
			110	0101	1110	0101					111	1111	1111	1111	
		110	0101	1110	0101					111	1111	1111	1111		
	110	0101	1110	0101					111	1111	1111	1111			
#5				0010	0010	0010	0010	#13				0011	0001	0011	0001
			100	0010	0010	0010					011	0001	0011	0001	
		100	0010	0010	0010					011	0001	0011	0001		
	100	0010	0010	0010					011	0001	0011	0001			
#6				0110	0110	0110	0110	#14				0001	0011	0001	0011
			100	0110	0110	0110					111	0011	0001	0011	
		100	0110	0110	0110					111	0011	0001	0011		
	100	0110	0110	0110					111	0011	0001	0011			
#7				1010	1010	1010	1010	#15				0001	0001	0001	0001
			100	1010	1010	1010					011	0111	0111	0111	
		100	1010	1010	1010					111	0001	0001	0001		
	100	1010	1010	1010					011	0111	0111	0111			
#8				1011	1011	1011	1011	#16				0111	0111	0111	0111
			010	1100	1100	1100					111	0001	0001	0001	
		101	1011	1011	1011					011	0111	0111	0111		
	010	1100	1100	1100					111	0001	0001	0001			

3.1.3. Design Evaluation.

In the design of MCPM, the cells are modified as follows. Each left-boundary cell only consists of a full adder, i.e., the corresponding AND gate is separated from the cell and this AND gate is connected to an XOR gate with a control signal. The remaining cells are designed in such a way that each cell retains the same operation as in the original design, but produces a logical 1 carry output when the input combination is (1000).

The extra hardware in an n-by-n MCPM are those (n-1)'s XOR gates. Unlike the XOR gates located at the critical path in the design of MCPM in [9], the XOR gates in the proposed MCPM design will not degrade the speed performance.

3.2. Carry-Save Array Multipliers (CSM)

A 4-by-4 Carry-Save array multiplier (CSM) is illustrated in Figure 11(a) [3]. It has been shown that the CSM is not C-testable [8]. Therefore, in this section, the design of C-testable CSM is studied and the graph labeling scheme is also applied to generate test patterns.

3.2.1. Graph Labeling

Consider the four basic cells, M_i, 1=1 to 4, as shown in Figure 11(b). According to the interconnection topology in Figure 11(b) and the mappings (3) and (4), we should solve the following mappings (12)-(15) for L_{ij}'s, $1 \le i,j \le 4$.

$$L_{11}L_{12}L_{13} \longrightarrow L_{32}L_{41} \; ; \tag{12}$$

$$L_{21}L_{22}L_{23} \longrightarrow L_{42}L_{31} \; ; \tag{13}$$

$$L_{31}L_{32}L_{33} \longrightarrow L_{12}L_{21} \; ; \text{ and} \tag{14}$$

$$L_{41}L_{42}L_{43} \longrightarrow L_{22}L_{11} \; . \tag{15}$$

Similar to Theorem 1, the following Theorem and Corollary result.

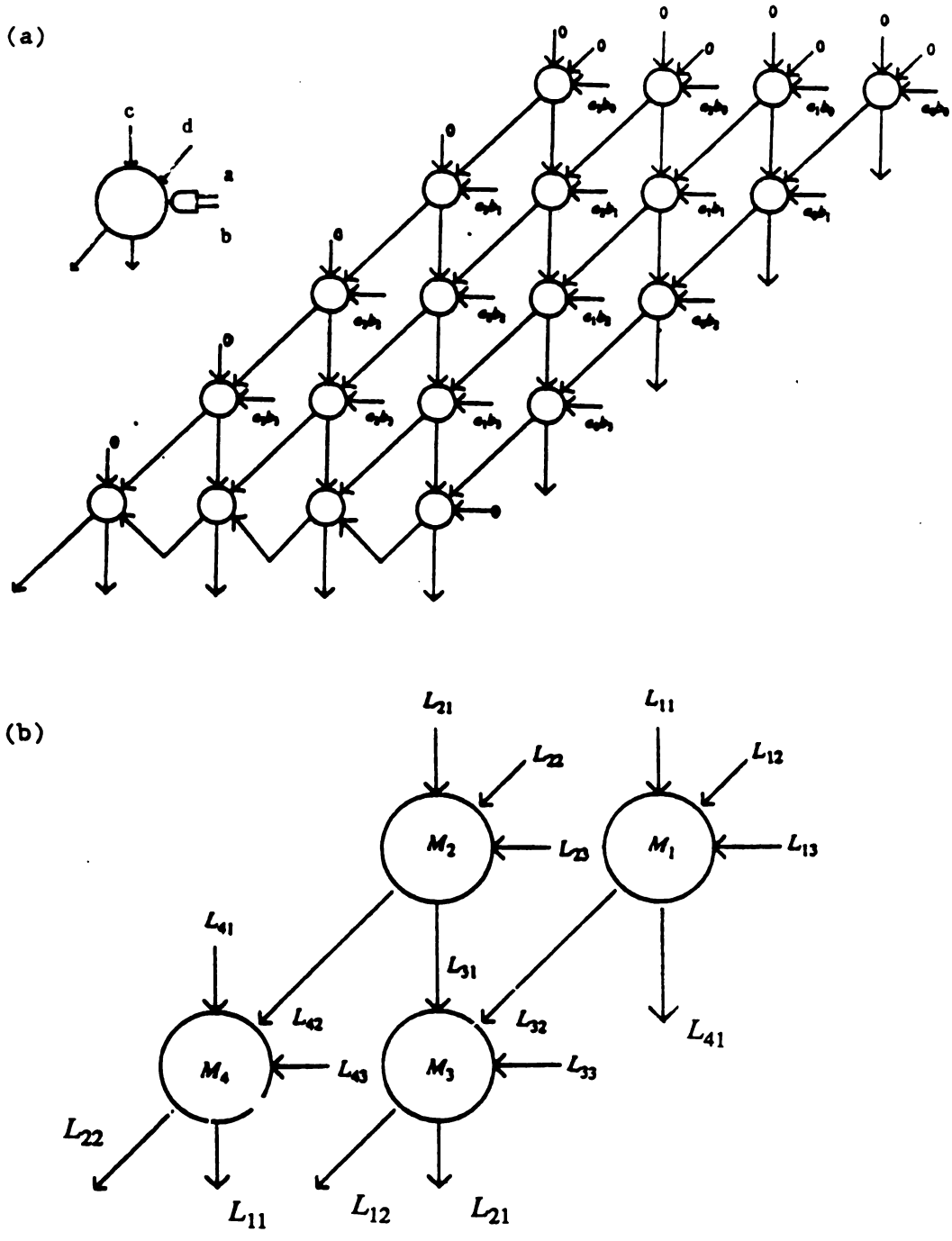


Figure 11. A 4-by-4 CSM : (a) Schematic Circuit Diagram;
(b) Four Basic Cells; (c) Labeling; and (d) Modified CSM.

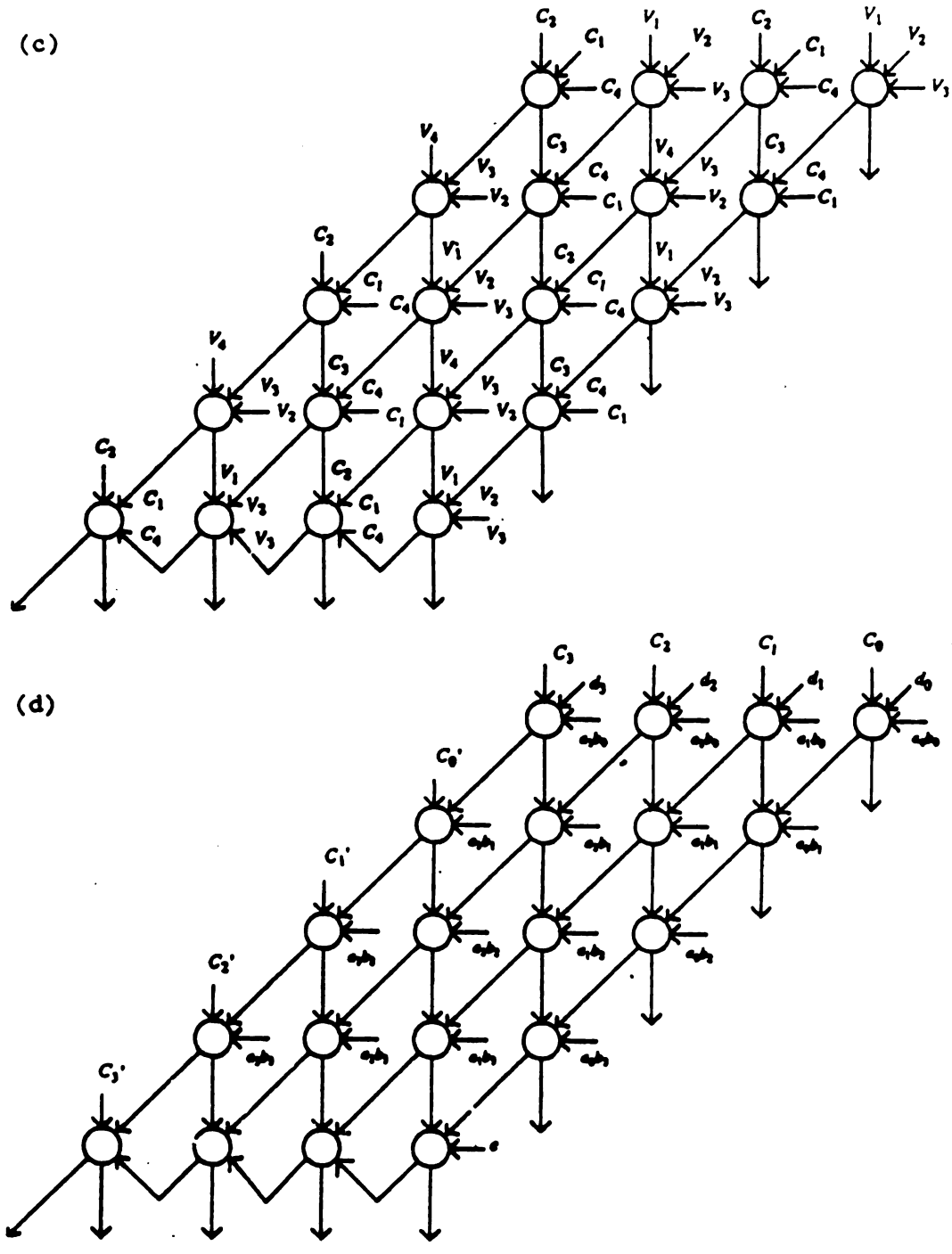


Figure 11. (Continued)

Theorem 3.

The following set of labels is a solution of the mappings (12)-(15) :

$$
\begin{aligned}
&L_1=(L_{11},L_{12},L_{13})=(V_1,V_2,V_3) \ ; \\
&L_2=(L_{21},L_{22},L_{23})=(C_2,C_1,C_4) \ ; \\
&L_3=(L_{31},L_{32},L_{33})=(C_3,C_4,C_1) \ ; \\
&L_4=(L_{41},L_{42},L_{43})=(V_4,V_3,V_2) \ .
\end{aligned}
\tag{16}
$$

Proof : Similar to the proof of Theorem 1, with the mappings (12)-(15), we can obtain the labels at (16).

Corollary 3.1.

There exist 24 possible sets of such labels.

Proof : Similar to Corollary 1.2, label L_1 takes three indices out from a set of four, this implies that there exist 12 possible sets. Furthermore, since L_1 can take either all V_i, or all C_i, this results in a total of 24 sets.

If the labels (16) are employed, they can be perfectly applied to the array without any extra hardware, as shown in Figure 11(c). Similar to the construction of Table 6, the application of such labels to CSM under the external constraints, is illustrated in Table 10.

Table 10. Application of L_i's to CSM.

Vector 1				Vector 2				Vector 3				Vector 4			
V_1	V_2	V_3	a_i b_j	C_2	C_1	C_4	a_i b_j	C_3	C_4	C_1	a_i b_j	V_4	V_3	V_2	a_i b_j
0	0	0	*	0	0	0	*	0	0	0	*	0	0	0	*
0	0	1	1 1	1	1	0	0 1	0	0	1	1 1	1	1	0	0 1
0	1	0	1 0	0	1	0	1 0	1	0	1	1 1	1	0	1	1 1
0	1	1	1 1	0	1	1	1 1	0	1	1	1 1	0	1	1	1 1
1	0	0	*	1	0	0	*	1	0	0	*	1	0	0	*
1	0	1	1 1	1	0	1	1 1	0	1	0	1 0	0	1	0	1 0
1	1	0	0 1	0	0	1	1 1	1	1	0	0 1	0	0	1	1 1
1	1	1	1 1	1	1	1	1 1	1	1	1	1 1	1	1	1	1 1

Like the design of MCPM, the carry output of (0100) can be changed to 1 in order to reproduce the patterns (1100) and (1110) internally because the input combinations (0100) and (0101) would never appear in the CSM. This can be proved by the following lemma.

Lemma 5.

The input vectors (0100) and (0101) can never appear at the input to any cell in the CSM.

Proof: By Lemma 2, the input vectors (1001) and (0001) can never appear in CPM. Since the input vector (c,d,a,b) in the CSM is equivalent to a vector (b,a,c,d) in the CPM, hence, by Lemma 2, (0101) and (0100) can never appear at the input to any cell in the CSM.

Similar to Table 7, the input combinations for a MCSM derived from Table 10 are shown in Table 11.

Table 11. Input Combinations for a MCSM.

Test_#	Vector 1 (c d a b)	Vector 2 (c d a b)	Vector 3 (c d a b)	Vector 4 (c d a b)
1.	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2.	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1
3.	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0
4.	0 0 1 1	1 1 0 1	0 0 1 1	1 1 0 1
5.	0 1 1 0	0 1 1 0	1 0 1 1	1 0 1 1
6.	0 1 1 1	0 1 1 1	0 1 1 1	0 1 1 1
7.	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0
8.	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1
9.	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0
10.	1 0 1 1	1 0 1 1	0 1 1 0	0 1 1 0
11.	1 1 0 1	0 0 1 1	1 1 0 1	0 0 1 1
12.	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
13.	0 1 0 0	1 1 0 0	0 1 0 0	1 1 0 0
14.	1 1 0 0	0 1 0 0	1 1 0 0	0 1 0 0
15.	0 1 0 0	1 1 1 0	0 1 0 0	1 1 1 0
16.	1 1 1 0	0 1 0 0	1 1 1 0	0 1 0 0

In order to apply the test vectors in Table 10, those terminals assigned a logical 0 in a CSM should become accessible. Figure 11(d) is a 4-by-4 modified CSM (MCSM). The following lemmas and theorem can be concluded.

Lemma 6.

The basic FA/AND cell of a CSM can be tested with 16 tests.

Proof: A 4-input combinational circuit can be examined by all its input patterns, i.e., 16 tests.

Lemma 7.

The basic CSM cells in Figure 11(b) can be tested with 16 tests.

Proof: From the generation of Table 11, each basic cell can be exhaustively tested by those 16 patterns.

Theorem 4.

The MCSM is C-testable with a test length of 16.

Proof: Similar to the proof of Theorem 2, if M_{BC} is defined as the four basic cells of CSM and TS is the 16 test vectors in Table 11, then an n-by-n MCSM can be partitioned into m^2 M_{BC}'s, where n=2m. Since, by Lemma 7, each M_{BC} can then be tested by the test set TS, an n-by-n MCSM can be tested by the same test set TS. So, the MCSM is C-testable with a test length of 16.

3.2.2. Test Pattern Generation.

Consider the 4-by-4 modified Carry-Save array multiplier (MCSM), as shown in Figure 11(d). The c', c, d, and e are connected to logical 0 during multiplication. However, it is assumed that during testing, these inputs are available as primary inputs to the array.

Similar to Algorithm 1, Algorithm 2 generates both test patterns and expected outputs for a MCSM from Table 11. Table 12 illustrates the test patterns and expected outputs for a 4-by-4 MCSM.

Algorithm 2:

```
{*  Vector_i=(ic,id,ia,ib), i=1,..,4.
 *          ic : c-input, id : d-input, ia : a-input, ib : b-input.
 *  The Test patterns to be generated are:
 *    a(i), b(i), c(i), c'(i), d(i), where i=0...n, e.  n is an odd number.
 *  The expected product is p(i), where i=0...2n+1.  *}

 {* Step 1. (Test Patterns)    *}

  For i=0 to n by 2
      do Begin
        a(i):=1a; a(i+1):=2a;
        b(i):=1b; b(i+1):=3b;
        c'(i):=4c; c'(i+1):=2c;
        c(i):=1c; c(i+1):=2c;
        d(i):=1d; d(i+1):=2d;
      End;
  e:=a(0)*b(0);

{* Step 2: (Expected Results)    *}

  For i=0 to n By 2
      do Begin
        p(i):=4c; p(i+1):=2c;
      End;
  For i=n+1 To 2n+1 By 2
      do Begin
        p(i):=4c; p(i+1):=3c;
      End;
  If (Test_#=13 OR Test_#=15) Then
        For i=n+2 To 2n+1 do     p(i):=p(i)';
  If (Test_#=14 OR Test_#=16) Then
        For i=n+1 To 2n+1 do     p(i):=p(i)';
```

Table 12. Test Patterns and Expected Outputs for MCSM.

Test #	a	b	c'	c	d	e	Expected Results
1.	0000	0000	0000	0000	0000	0	00000000
2.	0000	1111	0000	0000	0000	0	00000000
3.	1111	0000	0000	0000	0000	0	00000000
4.	0101	1111	1111	1010	1010	1	01011111
5.	1111	1010	0101	0000	1111	0	11110101
6.	1111	1111	0000	0000	1111	1	00000000
7.	0000	0000	1111	1111	0000	0	11111111
8.	0000	1111	1111	1111	0000	0	11111111
9.	1111	0000	1111	1111	0000	0	11111111
10.	1111	0101	1010	1111	0000	1	00001010
11.	1010	1111	0000	0101	0101	0	10100000
12.	1111	1111	1111	1111	1111	1	11111111
13.	0000	0000	1111	1010	1111	0	10011111
14.	0000	0000	0000	0101	1111	0	01000000
15.	1010	0000	1111	1010	1111	0	10011111
16.	0101	0000	0000	0101	1111	0	01000000

Remark : a=($a_3a_2a_1a_0$) b=($b_3b_2b_1b_0$) c=($C_3C_2C_1C_0$)
c'=($C_3'C_2'C_1'C_0'$) d=($d_3d_2d_1d_0$)

3.2.3. Design of an Alternative C-testable CSM.

Consider an alternative Carry-Save Array Multiplier for multiplying two 5-bit unsigned binary numbers, as shown in Figure 12(a) [4], referred to as CSM_B. The interconnection topology shows that each a_i is fed to the topmost cell and the cells in the next diagonal column. Each b_j is fed to the cells in a row and the leftmost cell of its next row. Under the external constrains, the CSM_B is modified as shown in Figure 12(b). Control signals, d's, S_1, S_2, S_3, and e, are for producing the sequence of input combinations in Table 10. During normal multiplication, the signals d's, S_1, S_2, S_3, and e are all set to logical 0. Each cell in the top row constructed by two AND gates and a full-adder is now modified by inserting an XOR gate between the AND gate and the basic cell as shown in Figure 12(b). Either signal S_1 or S_2 is XORed with the output of this AND gate. The signal S_1 (S_2) is applied to the odd (even) numbered cells of the top row. The left-boundary cells are modified by adding an XOR gate with two inputs: S_3 and b_j. In addition, a signal e=a_0b_1 is applied as the initial carry of the carry propagate stage. According to Table 11, Algorithm 3 generates both test patterns and expected outputs for a MCSM_B. Table 13 illustrates the generated test patterns and expected outputs for a 5-by-5 MCSM_B. Applying the test patterns of Table 13 to the MCSM_B allows us to conclude that the MCSM_B is C-testable with a test length of 16.

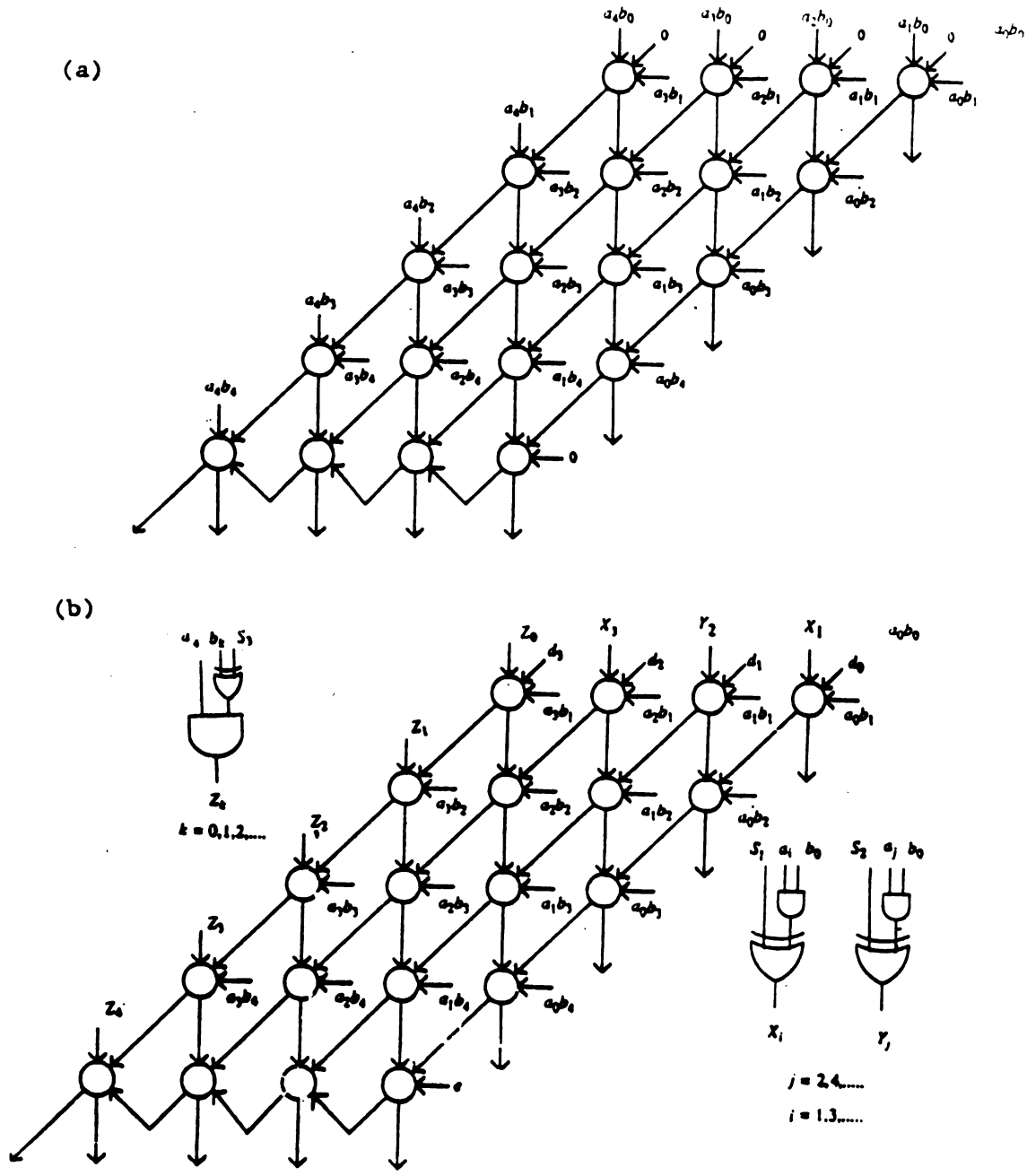


Figure 12. Schematic Circuit Diagram : (a) A 5-by-5 CSM_B [6]; and (b) A 5-by-5 Modified CSM_B.

Algorithm 3:

```
{*    Vector_i=(ic,id,ia,ib), i=1,..,4.
 *          ic : c-input, id : d-input, ia : a-input, ib : b-input.
 *    The test patterns to be generated are:
 *      a(i), b(i), d(i), where i=0...n, e, s1, s2, s3.
 *    The expected result is p(i), where i=1...2n+1.
 *    n is even.          *}

{* Step 1. (Test Patterns)   *}

  For i=0 to n by 2
      do Begin
        d(i):=1d; d(i+1):=2d;
        a(i):=1a; a(i+1):=2a;
        b(i+1):=1b; b(i+2):=3b;
      End;
  e:=a(0)*b(1);
  If (2c=1 OR 4c=1) Then a(n):=1  Else a(n):=0;
  If (1a=1 AND 2a=0) Then a(n):=1;
  For k=0 to 1
      do Begin
        b(0):=k;
        If (1a*b(0)=1c) Then s1:=0  Else s1:=1;
        If (2a*b(0)=1d) Then s2:=0  Else s2:=1;
        If (a(n)*b(1)=4c) Then s3:=0  Else s3:=1;
        If (a(n)*(b(0) XOR s3)=2c) Then k=1;
      End;
  If (1d=2d=1 AND 1a=0) Then b(0):=1;

{* Step 2: (Expected Results)   *}

  For i=0 to n By 2
      do Begin
        p(i):=4c; p(i+1):=2c;
      End;
  For i=n+1 to 2n+1 By 2
      do Begin
        p(i):=4c; p(i+1):=3c;
      End;
  If (Test_#=13 OR Test_#=15) Then
```

For i=n+2 To 2n do p(i):=$\overline{p(i)}$;
If (Test_#=14 OR Test_#=16) Then
For i=n+1 To 2n do p(i):=$\overline{p(i)}$;

Table 13. Test Patterns and Expected Outputs for a 5-by-5 MCSM_B.

Test #	a	b	d	S_1 S_2 S_3	Expected Results
1.	00000	00000	0000	0 0 0	0000000000
2.	00000	11110	0000	0 0 0	0000000000
3.	01111	00000	0000	0 0 0	0000000000
4.	10101	11111	1010	0 0 0	1010111110
5.	11111	10101	1111	1 1 1	0111101010
6.	01111	11110	1111	0 0 0	0000000000
7.	10000	00000	0000	1 1 1	0111111110
8.	10000	11111	0000	1 1 0	0111111110
9.	11111	00000	0000	1 1 1	0111111110
10.	11111	01010	0000	1 1 1	1000010100
11.	01010	11110	0101	1 0 0	0101000000
12.	11111	11111	1111	0 0 0	1111111110
13.	10000	00000	1111	0 1 1	1100111110
14.	00000	00001	1111	1 0 0	1010000000
15.	11010	00000	1111	0 1 1	1100111110
16.	10101	00000	1111	1 0 0	1010000000

Remark : a=($a_4a_3a_2a_1a_0$) b=($b_4b_3b_2b_1b_0$) d=($d_3d_2d_1d_0$)

Theorem 5.

The MCSM_B is C-testable with 16 test patterns.

Proof: The only difference between the MCSM_B and MCSM is in their primary inputs of the boundary cells. For simplicity of discussion, the gates that produce the outputs X_i, Y_j, and Z_k, as shown in Figure 12, are denoted to as cell $\mathbf{X}_i$, $\mathbf{Y}_j$, and $\mathbf{Z}_k$, respectively. Since the control signals S_1, S_2 and S_3 are set to

zero during the normal operation, the possible input combinations for the cells $\mathbf{X}_i$, $\mathbf{Y}_j$, and $\mathbf{Z}_k$ are:

for $\mathbf{X}_i$: (S_1,a_i,b_0)=(000),(001),(010),(011);

$\mathbf{Y}_j$: (S_2,a_j,b_0)=(000),(001),(010),(011);

$\mathbf{Z}_k$: (a_n,b_k,S_3)=(000),(010),(100),(110).

Since it has been shown that the MCSM is C-testable (Theorem 6), hence, the only problem remained is whether or not the above combinations can be included when the 16 patterns are applied. With the selected control signals, S_1, S_2, and S_3, and the application of the test set in Table 11, the following table illustrates the combinations applied in the cells $\mathbf{X}_i$, $\mathbf{Y}_j$ and $\mathbf{Z}_k$. The table shows that the above combinations are indeed included.

	$\mathbf{X}_i$	$\mathbf{Y}_j$	$\mathbf{Z}_0$	$\mathbf{Z}_{1,3,..}$	$\mathbf{Z}_{2,4,..}$
	$S_1\ a_i\ b_0$	$S_2\ a_j\ b_0$	$a_n\ b_0\ S_3$	$a_n\ b_k\ S_3$	$a_n\ b_k\ S_3$
1.	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
2.	0 0 0	0 0 0	0 0 0	0 1 0	0 1 0
3.	0 1 0	0 1 0	0 0 0	0 0 0	0 0 0
4.	0 0 1	0 1 1	1 1 0	1 1 0	1 1 0
5.	1 1 1	1 1 1	1 1 1	1 0 1	1 1 1
6.	0 1 0	0 1 0	0 0 0	0 1 0	0 1 0
7.	1 0 0	1 0 0	1 0 1	1 0 1	1 0 1
8.	1 0 1	1 0 1	1 1 0	1 1 0	1 1 0
9.	1 1 0	1 1 0	1 0 1	1 0 1	1 0 1
10.	1 1 0	1 1 0	1 0 1	1 1 1	1 0 1
11.	1 1 0	0 0 0	0 0 0	0 1 0	0 1 0
12.	0 1 1	0 1 1	1 1 0	1 1 0	1 1 0
13.	0 0 0	1 0 0	1 0 1	1 0 1	1 0 1
14.	1 0 1	0 0 1	0 1 0	0 0 0	0 0 0
15.	0 1 0	1 0 0	1 0 1	1 0 1	1 0 1
16.	1 0 0	0 1 0	1 0 0	1 0 0	1 0 0

3.3. Baugh-Wooley Array Multiplier (BWM)

A 5-by-5 Baugh-Wooley Array Multiplier (BWM) is illustrated in Figure 13. It has been shown that the BWM is not C-testable [8].

3.3.1. Design of C-testable MBWM

A MCSM_C can be constructed from a MCSM_B if the cells in the second row from the bottom, or the (n-1)th row of an n-by-n MCSM_B, is modified as shown in Figure 14, where each a_k is replaced by an XOR gate having two inputs a_k and S_4. When S_4 is 0, the MCSM_C is functionally equivalent to a MCSM_B. Therefore, the test patterns of Table 13 can applied all the input combinations in Table 11 to every cell of the array and detect any single fault in MCSM_C. However, in order to detect the possible faults in the added XOR gates, the control signal S_4 is assigned as shown in Table 14, where S_4 is set to logical 0 except for Test_#1, #3, #5 and Test_#8. With the application of the test patterns of Table 14, it can be concluded that MCSM_C is C-testable.

Lemma 8.

MCSM_C is C-testable with a test length of 16.

Proof: The MCSM_C is modified from MCSM_B as shown in Figure 14. Similar to the proof of Theorem 5, it is found that all the possible combinations of the

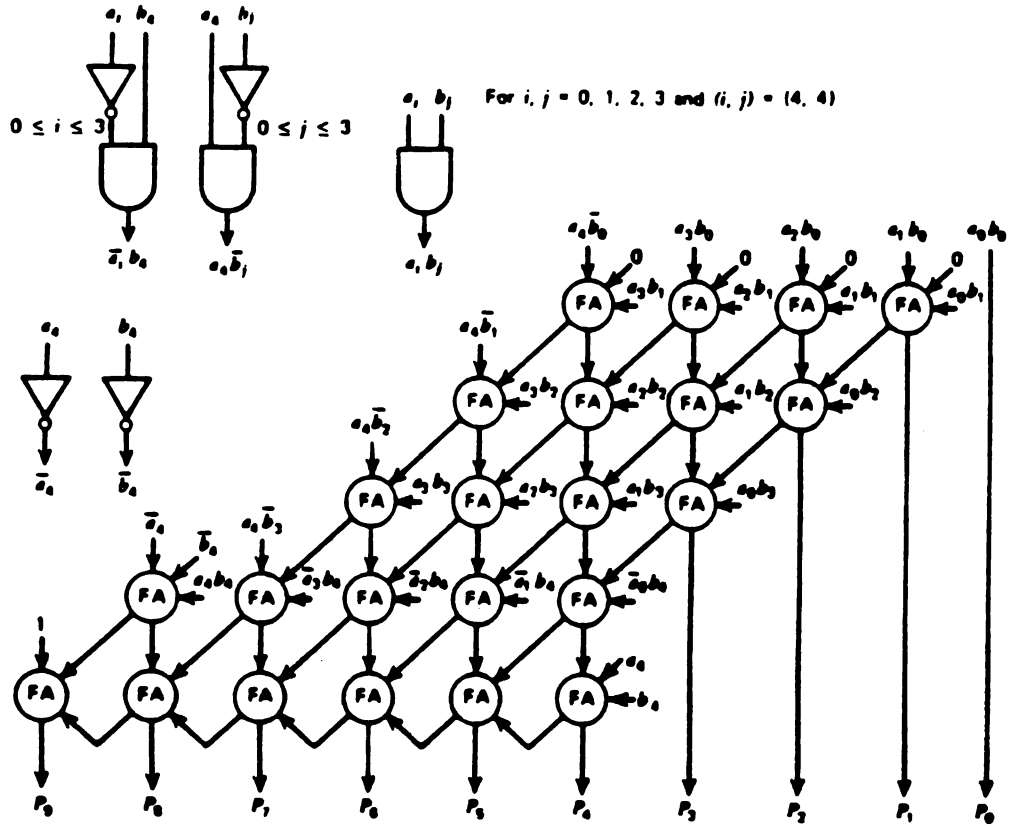


Figure 13. A Schematic Circuit Diagram of a 5-by-5 Baugh-Wooley Array Multiplier [6].

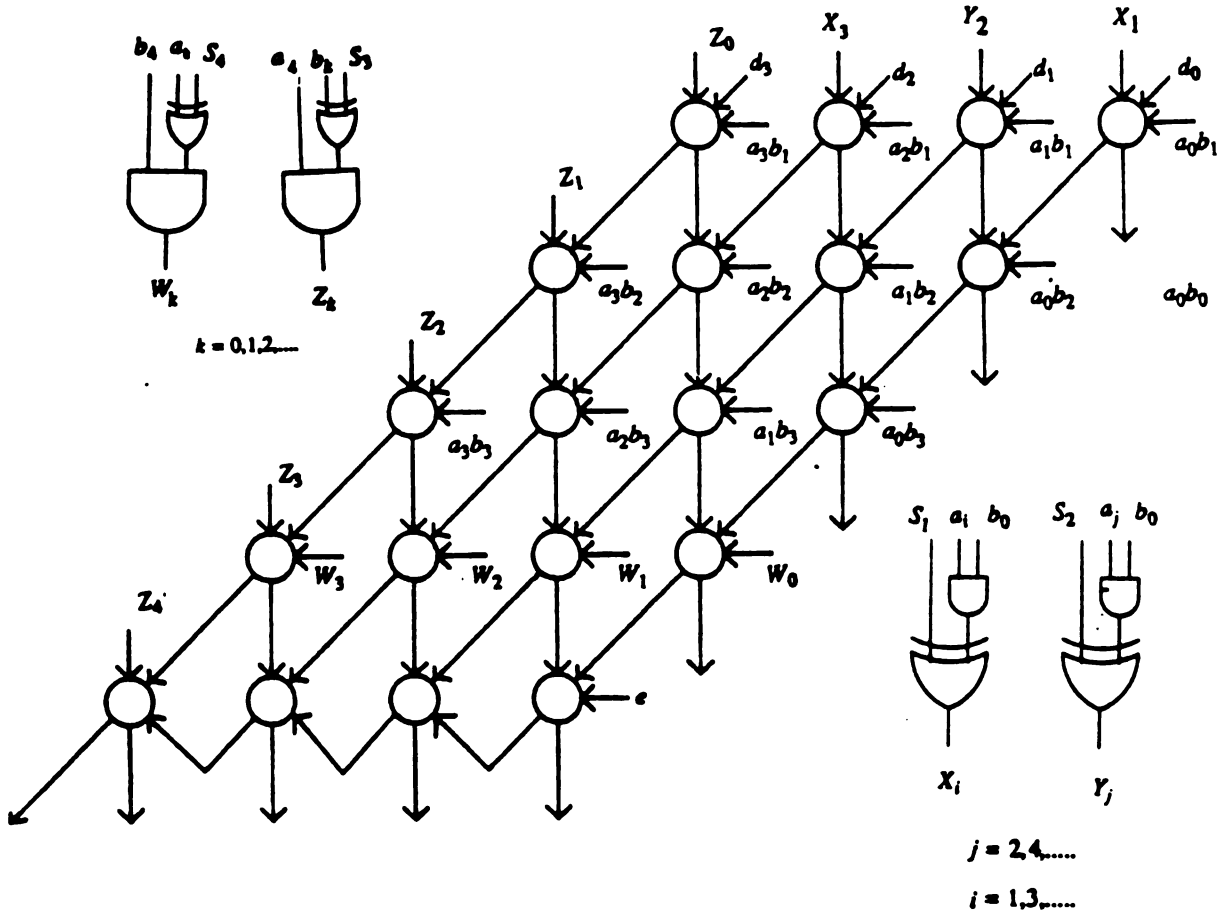


Figure 14. A Schematic Circuit Diagram of a 5-by-5 MCSM_C.

cells on W_k,

$$W_k : (b_4, a_k, S_4)=(001),(011),(101),(111),$$

are indeed included when the control signal S_4 is appropriately selected and applied. Since the MCSM_B is C-testable with a test length of 16, so is the MCSM_C.

Table 14. Test Patterns and Expected Outputs for a MCSM_C.

Test_#	a	b	d	S_1 S_2 S_3 S_4	Expected Results
1.	00000	00000	0000	0 0 0 1	0000000000
2.	00000	11111	0000	0 0 0 0	0000000000
3.	01111	00000	0000	0 0 0 1	0000000000
4.	10101	11111	1010	0 0 0 0	1010111111
5.	11111	10101	1111	1 1 1 1	1111101011
6.	01111	11110	1111	0 0 0 0	1000000000
7.	10000	00000	0000	1 1 1 0	0111111110
8.	10000	11111	0000	1 1 0 1	1111111110
9.	11111	00000	0000	1 1 1 0	0111111110
10.	11111	01010	0000	1 1 1 0	1000010100
11.	01010	11110	0101	1 0 0 0	0101000000
12.	11111	11111	1111	0 0 0 0	1111111111
13.	10000	00000	1111	0 1 1 0	1100111110
14.	00000	00000	1111	1 0 0 0	1010000000
15.	11010	00000	1111	0 1 1 0	1100111110
16.	00101	00000	1111	1 0 0 0	1010000000

Remark : a=$(a_4a_3a_2a_1a_0)$ b=$(b_4b_3b_2b_1b_0)$ d=$(d_3d_2d_1d_0)$

The MCSM_C can be further modified by adding extra cells, as shown in Figure 15. During the normal operation, the control signals d, S_1, S_2, S_3, S_5, and S_6 are

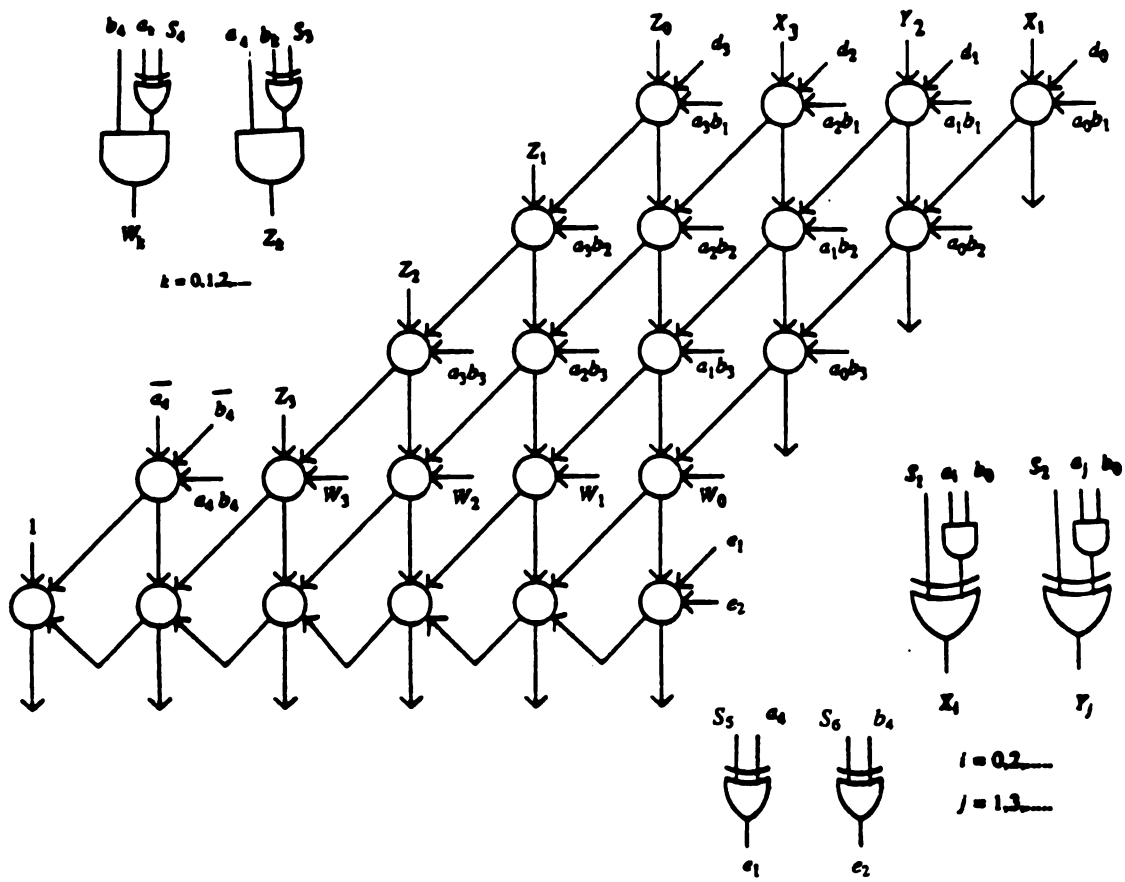


Figure 15. A schematic Circuit Diagram of a 5-by-5 MBWM.

set to logical 0 and the signal S_4 is set to logical 1. It is obvious that, with the above assignments, the circuit of Figure 15 is functionally equivalent to the Baugh-Wooley Array Multiplier of Figure 13. This circuit is referred to as modified BWM, or MBWM.

Theorem 6.

The MBWM is C-testable with a test length of 16.

Proof: The MBWM is modified from MCSM_C by adding extra cells as shown in Figure 15. With the appropriate selection of control signals, the 16 test pattern can be applied to test the extra cells. Therefore, the MBWM is C-testable with a test length of 16.

3.3.2. Test Pattern Generation.

The testing problem of the MBWM can be separated into two parts, one is for the testing of MCSM and the other is for those additional cells. In order to reduce the number of test patterns, one may overlap the testing of the two parts together. Algorithm 4 generates both test patterns and expected outputs for a MBWM from Table 11. The method used here is to apply the 16 test vectors in Table 11 into this array by controlling the primary inputs. Also, the input patterns should cover all the possible input combinations of the boundary cells. Table 15 illustrates the test patterns and expected outputs for a 5-by-5 MBWM.

Algorithm 4 :

```
{*  Vector_i=(ic,id,ia,ib), i=1,..,4.
 *        ic : c-input, id : d-input, ia : a-input, ib : b-input.
 *  The Test patterns to be generated are:
 *    a(i), b(i), d(i), i=0,...,n, (s6,..,s1)
 *  The expected product is p(i), i=0,...,2n+1.
 *  The function inv(arg) returns a value which is the complement of "arg".
 *  n is even.          *}

{*  Step 1. (Test Patterns)    *}

  For i=0 to n by 2
      do Begin
        a(i):=1a; a(i+1):=2a;
        b(i+1):=1b; b(i+2):=3b;
        d(i):=1d; d(i+1):=2d;
      End;

  If (2c=1 OR 4c=1) Then a(n):=1 Else a(n):=0;
  If (1a=1 AND 2a=0) Then a(n):=1;
  For k=0 to 1
      do Begin
        b(0):=k;
        If (a(1)*b(0)=c1) Then s1:=0  Else s1:=1;
        If (a(2)*b(0)=c2) Then s2:=0  Else s2:=1;
        If (a(n)*b(1)=c4) Then s3:=0  Else s3:=1;
        If (a(n)*(b(0) XOR s3)=2c) Then k=1;
      End;
  If (1d=2d=1 AND 1a=0 ) Then b(0):=1;
  s4:= b(n)' * s1' * s2' * s3';
  If (d2≠a(n)) Then s5:=1  Else s5:=0;
  If (a2*b2 ≠ b(n)) Then s6:=1  Else s6:=0;
  If (s1*s2*s3=1 AND s5=0) Then s4:=1;
  If (s1*s2*s5=1 AND s3=0) Then s4:=1;
```

```
{* Step 2: (Expected Results) *}

  p(0):=a(0)*b(0);
  For i=1 to n-1 by 2
       do Begin
         p(i):=4c; p(i+1):=2c;
       End;
  For i=n to 2n by 2
       do Begin
         p(i):=3c; p(i+1):=4c;
       End;
  If (a(n)≠2c OR b(n)≠2c) Then p(2n):=p(2n)‾;
  If (d2*a2*b2=1) Then
         p(2n+1):= inv(a(n))*inv(b(n))
       Else
         If (2d=0 AND 2a*2b=0) Then
               p(2n+1):= inv(inv(a(n))*inv(b(n)))
             Else
               p(2n+1):=0;
  If (Test_#=14,16) Then  For i=0 to n-1 do p(n+1+i):= p(n+1+i)‾;
  If (Test_#=13,15) Then  For i=0 to n-1 do p(n+2+i):= p(n+2+i)‾;

  If (s4=1 AND s5=0) Then
       Begin
         For i=n to 2n+1 Do  p(i):=0;
       End;
  If (s4=1 AND s5=1) Then
       Begin
         p(n):=p(n)‾;
         p(2n):=p(2n)‾;
         p(2n+1):=p(2n+1)‾;
       End;
```

Table 15. Test Patterns and Expected Outputs for a 5-by-5 MBWM.

Test #	a	b	d	S_1 - S_6	Expected Results
1.	00000	00000	0000	0 0 0 1 0 0	0000000000
2.	00000	11110	0000	0 0 0 0 0 1	1100000000
3.	01111	00000	0000	0 0 0 1 0 0	0000000000
4.	10101	11111	1010	0 0 0 0 0 1	0010101111
5.	11111	10101	1111	1 1 1 1 0 1	0000001011
6.	01111	11110	1111	0 0 0 0 1 0	0100000000
7.	10000	00000	0000	1 1 1 0 1 0	1111111110
8.	10000	11111	0000	1 1 0 1 1 1	0011101110
9.	11111	00000	0000	1 1 1 0 1 0	1111111110
10.	11111	01010	0000	1 1 1 0 1 1	0000000100
11.	01010	11110	0101	1 0 0 0 0 0	0001010000
12.	11111	11111	1111	0 0 0 0 0 0	0111111111
13.	10000	00000	1111	0 1 1 0 0 0	0101001110
14.	00000	00001	1111	1 0 0 0 1 0	1010010000
15.	11010	00000	1111	0 1 1 0 0 0	0101001110
16.	10101	00000	1111	1 0 0 0 1 0	1010010000

Remark : a=($a_4a_3a_2a_1a_0$) b=($b_4b_3b_2b_1b_0$) d=($d_3d_2d_1d_0$)

Table 16 illustrates the input combinations for each cell in a 5-by-5 MBWM. It is obvious that, similar to the design of MCSM, all possible input combinations have been applied to each cell. However, the following combinations are not applied.

Cell 1 : (000), (011), (101), (111) ;

Cell 2 : (000), (001), (010), (011) ;

Cell 3 : (010), (001).

where Cell 1, Cell 2, and Cell 3 are the cells labeled in the Figure 16.

Table 16. Input Combinations for each cell in a 5-by-5 MBWM.

```
#1                                  0000 0000 0000 0000
                               0000 0000 0000 0000
                          0000 0000 0000 0000
               110 0010 0010 0010 0010
110            000  000  000  000  000

#2                                  0001 0001 0001 0001
                               0001 0001 0001 0001
                          0001 0001 0001 0001
               100 0001 0001 0001 0001
100            100  000  000  000  000

#3                                  0010 0010 0010 0010
                               0010 0010 0010 0010
                          0010 0010 0010 0010
               110 0000 0000 0000 0000
110            000  000  000  000  000

#4                                  1101 0011 1101 0011
                               1101 0011 1101 0011
                          1101 0011 1101 0011
               001 1101 0011 1101 0011
101            110  001  110  001  110

#5                                  0110 0110 0110 0110
                               1011 1011 1011 1011
                          0110 0110 0110 0110
               001 1001 1001 1001 1001
101            101  101  101  101  110

#6                                  0111 0111 0111 0111
                               0111 0111 0111 0111
                          0111 0111 0111 0111
               100 0111 0111 0111 0111
101            111  011  011  011  011

#7                                  1000 1000 1000 1000
                               1000 1000 1000 1000
                          1000 1000 1000 1000
               010 1000 1000 1000 1000
100            100  100  100  100  100

#8                                  1001 1001 1001 1001
                               1001 1001 1001 1001
                          1001 1001 1001 1001
               001 1011 1011 1011 1011
101            110  010  010  010  000

#9                                  1010 1010 1010 1010
                               1010 1010 1010 1010
                          1010 1010 1010 1010
               010 1010 1010 1010 1010
100            100  100  100  100  100

#10                                 1011 1011 1011 1011
                               0110 0110 0110 0110
                          1011 1011 1011 1011
               010 0110 0110 0110 0110
101            101  101  101  101  101

#11                                 0011 1101 0011 1101
                               0011 1101 0011 1101
                          0011 1101 0011 1101
               100 0011 1101 0011 1101
101            101  110  001  110  001

#12                                 1111 1111 1111 1111
                               1111 1111 1111 1111
                          1111 1111 1111 1111
               001 1111 1111 1111 1111
101            111  111  111  111  111

#13                                 1100 0100 1100 0100
                               1100 0100 1100 0100
                          1100 0100 1100 0100
               010 1100 0100 1100 0100
101            111  011  111  011  110

#14                                 0100 1100 0100 1100
                               0100 1100 0100 1100
                          0100 1100 0100 1100
               110 0100 1100 0100 1100
111            011  111  011  110  010

#15                                 1110 0100 1110 0100
                               1110 0100 1110 0100
                          1110 0100 1110 0100
               010 1110 0100 1110 0100
101            111  011  111  011  110

#16                                 0100 1110 0100 1110
                               0100 1110 0100 1110
                          0100 1110 0100 1110
               110 0100 1110 0100 1110
111            011  111  011  110  010
```

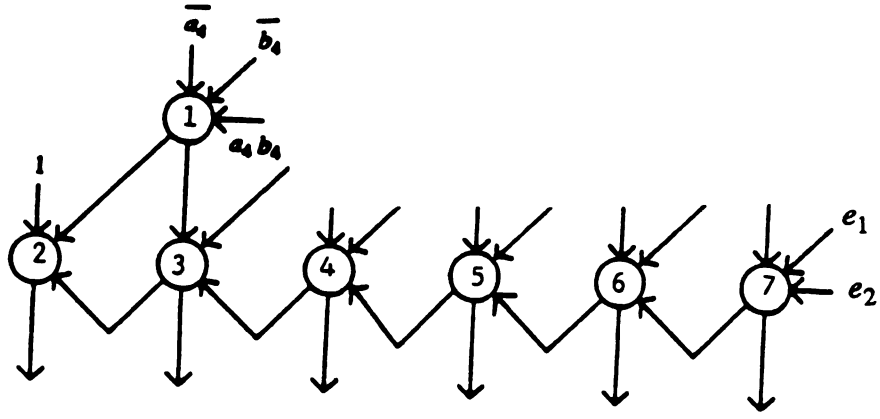


Figure 16. The Bottom Row of MBWM in Figure 15.

Consider all the possible combinations of a_4 and b_4.

a_4	b_4	Input Combinations
0	0	1 1 0
0	1	1 0 0
1	0	0 1 0
1	1	0 0 1

Because of the external constrain, all the possible input combinations are demonstrated in the above table. Therefore, input combinations (000), (011), (101), and (111) do not occur in Cell 1 during the normal operation.

Since the top input of Cell 2 is always 1 during normal operation, this implies that input combinations (000), (001), (010), and (011) do not occur in Cell 2 during the normal operation.

Finally, consider the interconnection of Cell 3 as shown in Figure 16. Also consider the following truth table for the sum bit of Cell 1.

a_4	b_4	sum
0	0	0
0	1	1
1	0	1
1	1	1

In other words, both a_4 and b_4 must be zeros if the sum bit of the Cell 1 is zero. Suppose that (010) can occur in Cell 3, i.e. the top input of Cell 3, or the sum output of Cell 1, is logical 0. This results in both a_4 and b_4 being 0. Therefore, the inputs Z_3 and W_3 of Figure 15 are also logical 0 and the cell would never produce a logical 1 in its carry-out bit. This implies that (010) does not occur in Cell 3 during normal operation.

Similarly, it can be easily found that (001) will never occur in Cell 3.

The above arguments show that the possible input combinations of those extra cell are indeed included when the 16 test patterns are applied.

IV. Design of C-testable Array Dividers

4.1. Non-Restoring Array Divider.

A 4-by-4 Non-restoring array divider (NRD) is illustrated in Figure 17. This divider receives a 7-bit dividend, 4-bit divisor, and produces a 4-bit quotient and a 4-bit remainder. In this section, the design of C-testable NRD is presented. The graph labeling scheme is also applied to generate test patterns.

The basic building block of a non-restoring divider is a controllable adder/subtractor (CAS) [6], as shown in Figure 18, where

$$S=X\oplus Y\oplus Z\oplus D, \text{ and}$$

$$P=(Y\oplus D)X+(Y\oplus D)Z+XZ.$$

When D=0, the cell is merely a full adder, i.e., S is the sum of X, Y, and Z and P is the carry. On the other hand, when D=1, the cell becomes a full adder with inputs X, Z and $\overline{Y}$. Consequently, the labeling scheme developed for full-adder-based array multipliers is also suitable for NRD.

4.1.1. Graph Labeling.

Because of the regularity of the NRD, the four basic CAS cells, M_i, i=1 to 4, as shown in Figure 19 can be found. According to the interconnection topology of the

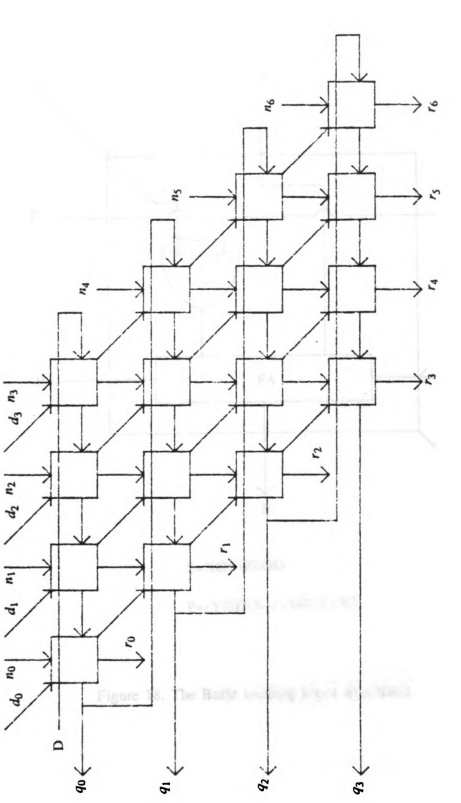
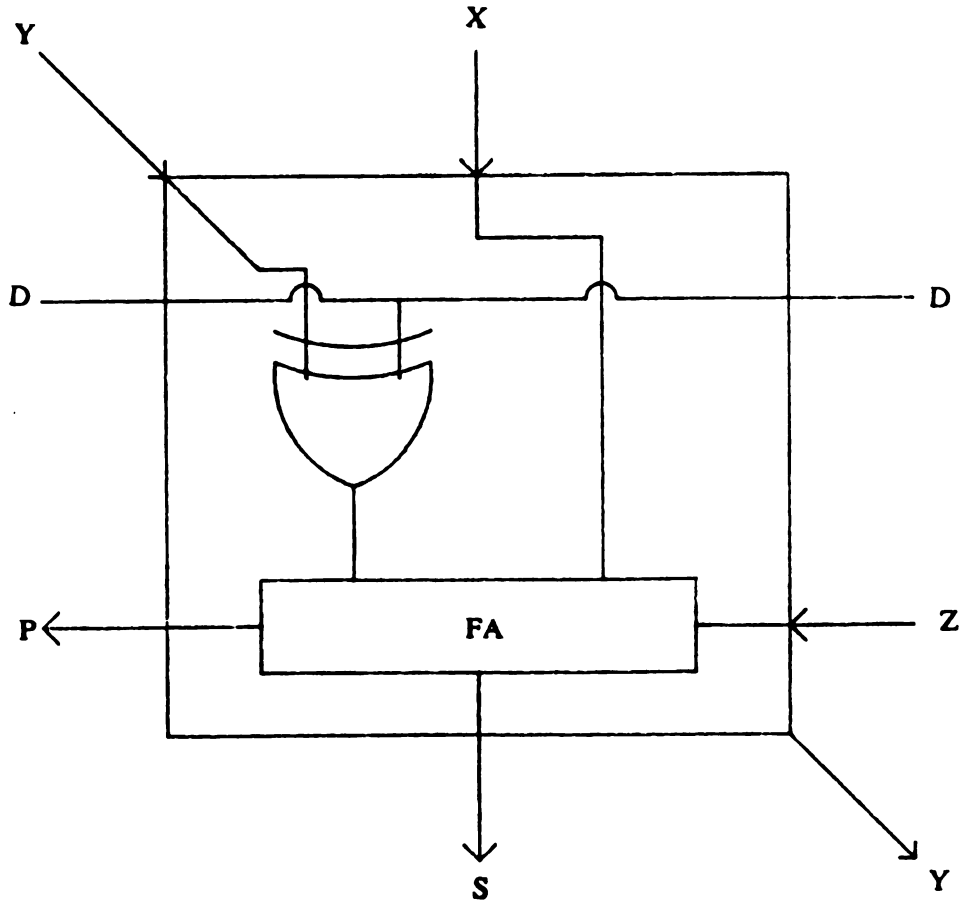


Figure 17. A Schematic Circuit Diagram of a 4-by-4 NRD [9].



$$S=X\oplus Y\oplus Z\oplus D$$

$$P=(Y\oplus D)X+(Y\oplus D)Z+XZ$$

Figure 18. The Basic building block of a NRD.

four basic cells, the mappings

$$
\begin{aligned}
&L_{11}L_{12}L_{13} \longrightarrow L_{23}L_{32} \ ; \\
&L_{21}L_{22}L_{23} \longrightarrow L_{13}L_{42} \ ; \\
&L_{31}L_{32}L_{33} \text{ --> } L_{43}L_{12} \ ; \\
&L_{41}L_{42}L_{43} \text{ --> } L_{33}L_{22} \ ; \\
&\text{where } L_{31}=L_{21},\ L_{41}=L_{11} \ ,
\end{aligned}
\tag{17}
$$

can be obtained. we shall solve the mappings for L_{ij}'s, i=1,2,3,4 and j=1,2,3, where i indicates the cell type and j represents the input.

Theorem 7.

The following set of labels is a solution of mapping (17) :

$$
\begin{aligned}
&L_1=(L_{11},L_{12},L_{13})=(V_1 \oplus D,V_2,V_3) \ ; \\
&L_2=(L_{21},L_{22},L_{23})=(C_1 \oplus D,C_2,C_4) \ ; \\
&L_3=(L_{31},L_{32},L_{33})=(C_1 \oplus D,V_4,C_3) \ ; \text{ and} \\
&L_4=(L_{41},L_{42},L_{43})=(V_1 \oplus D,C_3,V_4).
\end{aligned}
\tag{18}
$$

Proof : Similar to the proof of Theorem 1, with the mappings (17), we can solve for the labels in (18).

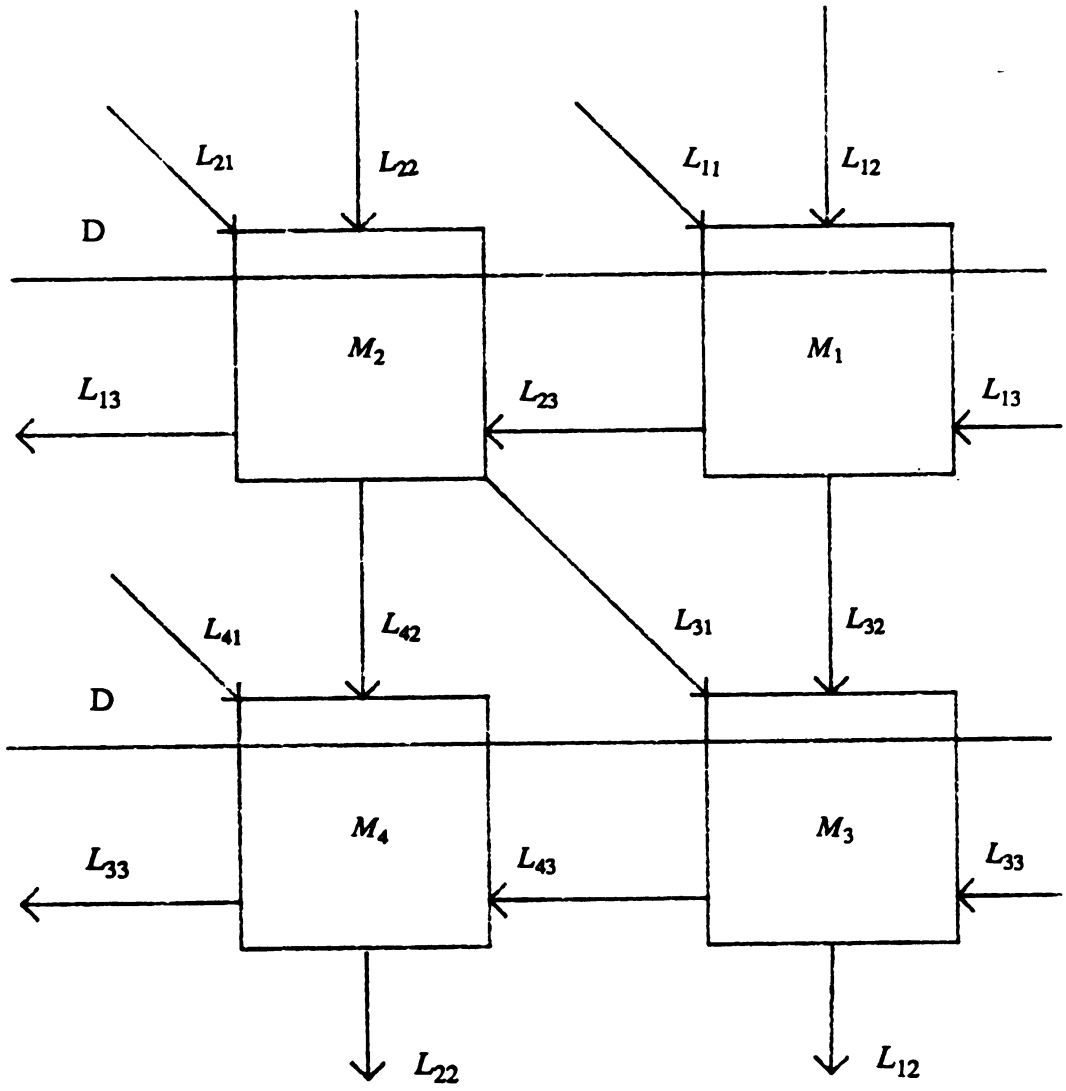


Figure 19. Four Basic Cells of a NRD.

Figure 20 shows the application of the labels in (18) to a 4-by-4 NRD. In Table 17, the test#1-8 and test#11-18 illustrate the input combinations generated directly from the label set (18). They contain all possible input combinations except the patterns { (010),(101) } (for D=0) and { (001),(110) } (for D=1) in both L_3 and L_4. Therefore, the tests #9, #10, #19, and #20 are added.

Tabel 17. Input combinations for an MNRD.

Test #	D	L_1 (y x z)	L_2 (y x z)	L_3 (y x z)	L_4 (y x z)
1.	0	0 0 0	0 0 0	0 0 0	0 0 0
2.	0	0 0 1	1 1 0	1 1 0	0 0 1
3.	0	0 1 0	1 0 0	1 1 1	0 1 1
4.	0	0 1 1	1 0 1	1 0 0	0 0 0
5.	0	1 0 0	0 1 0	0 1 1	1 1 1
6.	0	1 0 1	0 1 1	0 0 0	1 0 0
7.	0	1 1 0	0 0 1	0 0 1	1 0 0
8.	0	1 1 1	1 1 1	1 1 1	1 1 1
9.	0	0 1 0	0 1 0	0 1 0	0 1 0
10.	0	1 0 1	1 0 1	1 0 1	1 0 1
11.	1	1 0 0	1 0 0	1 0 0	1 0 0
12.	1	1 0 1	0 1 0	0 1 0	1 0 1
13.	1	1 1 0	0 0 0	0 1 1	1 1 1
14.	1	1 1 1	0 0 1	0 0 0	1 0 0
15.	1	0 0 0	1 1 0	1 1 1	0 1 1
16.	1	0 0 1	1 1 1	1 0 0	0 0 0
17.	1	0 1 0	1 0 1	1 0 1	0 0 0
18.	1	0 1 1	0 1 1	0 1 1	0 1 1
19.	1	0 0 1	0 0 1	0 0 1	0 0 1
20.	1	1 1 0	1 1 0	1 1 0	1 1 0

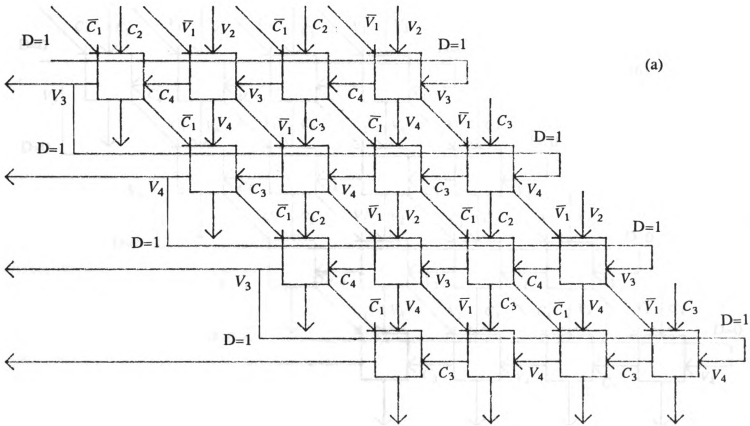


Figure 20. The application of labels to a NRD.
(a) D=1 (b) D=0

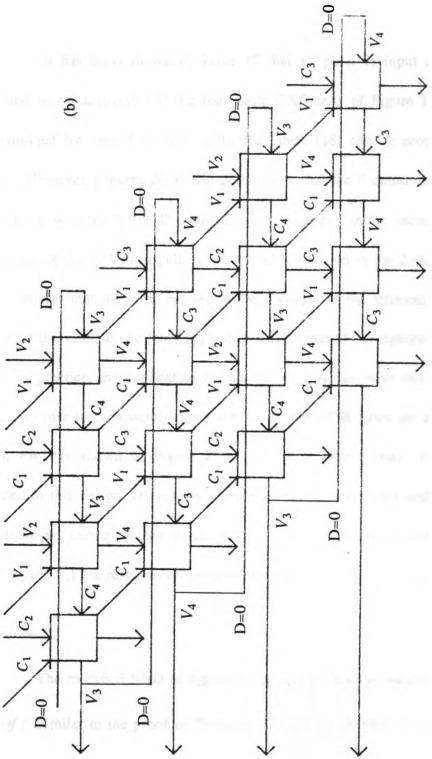


Figure 20. (Continued)

4.1.2. Test Pattern Generation.

It has been shown in Table 17 that all possible input combinations can be applied to exhaustively test the four basic CAS cells of Figure 18. Since an NRD is constructed by these four basic cells, the labels (18) can be propagated to the entire NRD. However, Figures 20(a) and 20(b) show that the P-output of the leftmost cell in the first row is fed to the D-input of the leftmost cell in the second row, and then the D-output of the rightmost cell in the second row is fed to the Z-input of the same cell. This implies that the label applied to the P-output of the leftmost cell in the first row must be the same as the label applied to the Z-input of the rightmost cell in the second row. In practice, the application of labels (18) may not meet this requirement. Therefore, in order to consistently apply the labels two XOR gates are added to the NRD in each row, as shown in Figure 21. Two extra control lines, Test1 and Test2, are needed in this design. During the normal operation, both Test1 and Test2 are at logical 0. However, during the test mode, if the size of the divider is even, labels V_3 (or $\overline{V}_3$) and V_4 (or $\overline{V}_4$) are respectively assigned to Test1 and Test2 for D=0 (or D=1).

Theorem 8.

The modified NRD of Figure 21 is C-testable with a test length of 20.

Proof: Similar to the proof of Theorem 2, if M_{BC} is defined as the four basic cells of NRD and TS is the 20 test vectors in Table 17, then an n-by-n MNRD can be partitioned into m^2 M_{BC}'s, where n=2m. Since each M_{BC} can be tested by the

test set TS, an n-by-n MNRD can then be tested by the same test set TS. So, the MNRD is C-testable with a test length of 20.

According to input combinations of Table 17, Algorithm 5 generates the test patterns and expected outputs for a MNRD. Table 18 shows the test patterns and expected outputs for a 4-by-4 MNRD.

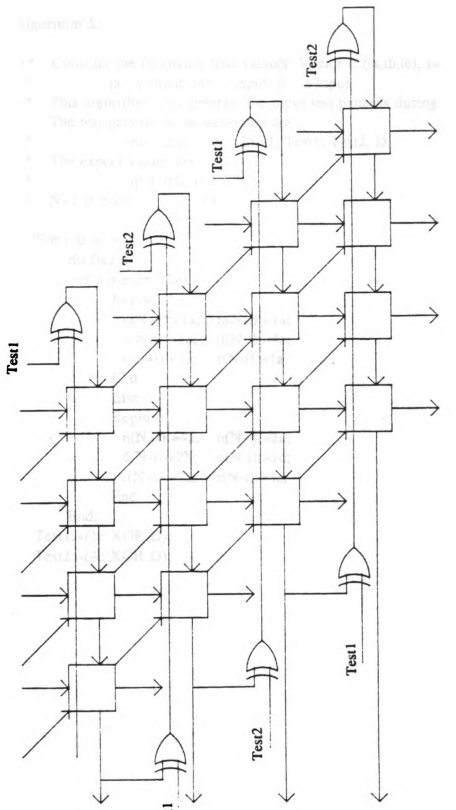


Figure 21. A Schematic Circuit Diagram of C-testable MNRD.

Algorithm 5:

```
{*  Consider the following four vectors: Vector i: (ia,ib,ic), i=1....4.
 *          ia : y-input, ib : x-input, ic : z-input.
 *  This algorithm is to generate the input test patterns during test mode.
 *  The test patterns to be generated are :
 *              n(i), d(i), i=0,...,2N+1, Test1, Test2, D
 *  The expect values are :
 *              q(i), r(i), i=0,...,N
 *  N+1 is even             *}

  For i=0 to N
      do Begin
        If i is even Then
            Begin
              n(N+i):=1a;   n(N-i):=1a;
              d(N-i):=1b;   q(N-i):=4c;
              r(N+i):=1a;   r(N-i):=1a;
            End
            Else
            Begin
              n(N+i):=4a;   n(N-i):=2a;
              d(N-i):=2b;   q(N-i):=1c;
              r(N+i):=2a;   r(N-i):=4a;
            End
      End;
  Test1:=(1c XOR D);
  Test2:=(4c XOR D);
```

Table 18. Test Patterns and Expected Outputs for a 4-by-4 MNRD.

Test#	n	d	Test1	Test2	r	q
1.	0000000	0000	0	0	0000000	0000
2.	1010000	1010	1	1	0000101	1111
3.	0101111	1010	0	1	1111010	0101
4.	0101010	1010	1	0	0101010	1010
5.	1010101	0101	0	1	1010101	0101
6.	1010000	0101	1	0	0000101	1010
7.	0101010	0101	0	0	0101010	0000
8.	1111111	1111	1	1	1111111	1111
9.	1111111	0000	0	0	1111111	0000
10.	0000000	1111	1	1	0000000	1111
11.	0000000	1111	1	1	0000000	0000
12.	1010000	1111	0	0	0000101	1111
13.	0101111	0101	1	0	1111010	0101
14.	0101010	0101	0	1	0101010	1010
15.	1010101	1010	1	0	1010101	0101
16.	1010000	1010	0	1	0000101	1010
17.	0101010	1010	1	1	0101010	0000
18.	1111111	0000	0	0	1111111	1111
19.	0000000	0000	0	0	0000000	1111
20.	1111111	1111	1	1	1111111	0000

Remark : $n=(n_0\ n_1\ n_2\ n_3\ n_4\ n_5\ n_6)$ $d=(d_0\ d_1\ d_2\ d_3)$
$r=(r_0\ r_1\ r_2\ r_3\ r_4\ r_5\ r_6)$ $q=(q_0\ q_1\ q_2\ q_3)$

4.2. Restoring Array Divider.

A Restoring Array Divider (RSD), as shown in Figure 22, is constructed by the identical building blocks, controllable subtractors (CSs) in Figure 23. The functions of each basic building block are

$$S=(X\oplus Y\oplus Z)\overline{D}+XD\ ;$$

and $$P=YZ+\overline{X}Z+\overline{X}Y\ .$$

It is obvious that the RSD is not C-testable because when D=1, S=X, any fault that occurs at either inputs Y or Z cannot be propagated to its output. In order to propagate the fault, the basic CS cell is modified as shown in Figure 24, where the cell functions become

$$S=(X\oplus Y\oplus Z)\overline{D}+XD\ ,$$

$$P=YZ+\overline{X}Z+\overline{X}Y\ , \qquad (19)$$

and $$B=A\oplus Y\oplus Z\ .$$

The modified CS (MCS) cell of Figure 24 receives an extra input A and produces an extra output B. Any fault that occurs at either input Y or Z can be propagated to the output B.

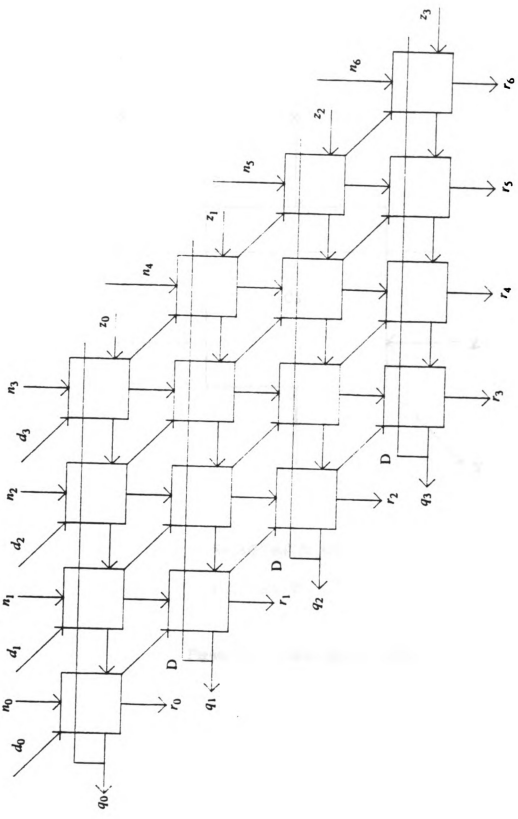
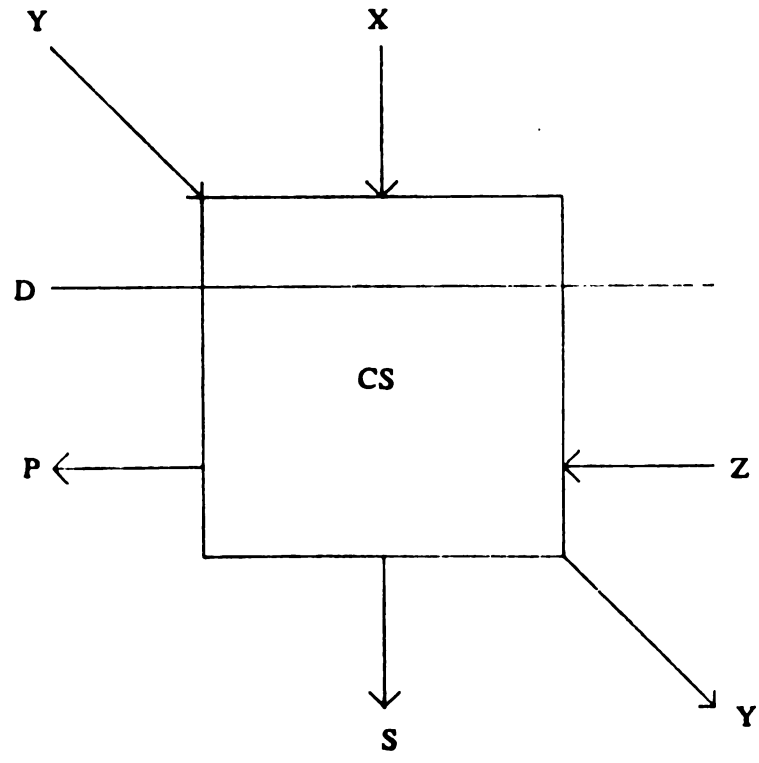


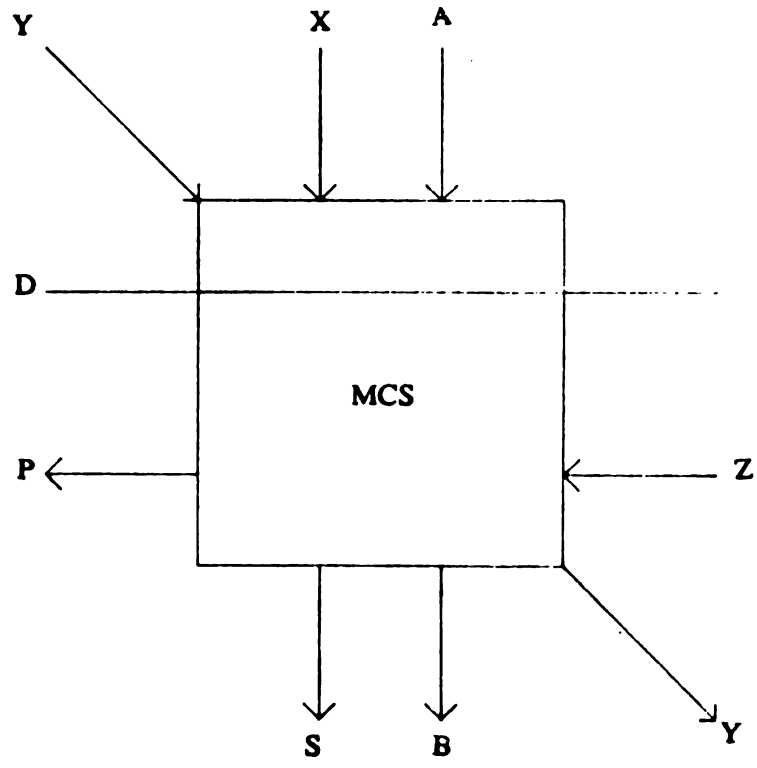
Figure 22. A Schematic Circuit Diagram of Restoring Array Divider.



$$S=(X\oplus Y\oplus Z)\overline{D}+XD$$

$$P=YZ+\overline{X}Z+\overline{X}Y$$

Figure 23. A Basic Cell of a RSD.



$S=X \oplus Y \oplus Z \oplus D$

$P=(Y \oplus D)X+(Y \oplus D)Z+XZ$

$B=A \oplus Y \oplus Z$

Figure 24. The Modified CS Cell of a MRSD.

4.2.1. Graph Labeling.

The MCS cell functions (19) can be described by the following mapping :

YXAZD ----> PSB, (20)

where P=f($\overline{X}$, Y, Z),

S=$\overline{D}$g(X, Y, Z)+DX,

and B=g(A, Y, Z).

Consider the case of D=1, the mapping (20) can be simplified as

YXAZ ----> PSB, (21)

where P=f($\overline{X}$, Y, Z),

S=X,

and B=g(A, Y, Z).

The function of f and g have been defined in (1) and (2).

Consider also the eight MCS cells of Figure 25. Each cell is labeled by $L_i=(L_{i1},L_{i2},L_{i3},L_{i4})$. Based on the topological interconnection of these cells, we shall solve for L_{ij}'s from the following mappings :

$$
\begin{aligned}
&L_{11}L_{12}L_{13}L_{14} \text{ ----> } L_{84}L_{52}L_{53},\\
&L_{21}L_{22}L_{23}L_{24} \text{ ----> } L_{74}L_{82}L_{83},\\
&L_{31}L_{32}L_{33}L_{34} \text{ ----> } L_{64}L_{72}L_{73},\\
&L_{41}L_{42}L_{43}L_{44} \text{ ----> } L_{54}L_{62}L_{63},\\
&L_{51}L_{52}L_{53}L_{54} \text{ ----> } L_{44}L_{32}L_{33},\\
&L_{61}L_{62}L_{63}L_{64} \text{ ----> } L_{34}L_{22}L_{23},\\
&L_{71}L_{72}L_{73}L_{74} \text{ ----> } L_{24}L_{12}L_{13},\\
&L_{81}L_{82}L_{83}L_{84} \text{ ----> } L_{14}L_{42}L_{43},\\
&\text{and } L_{11}=L_{21}=L_{31}=L_{41},\\
&\quad L_{51}=L_{61}=L_{71}=L_{81}.
\end{aligned}
\tag{22}
$$

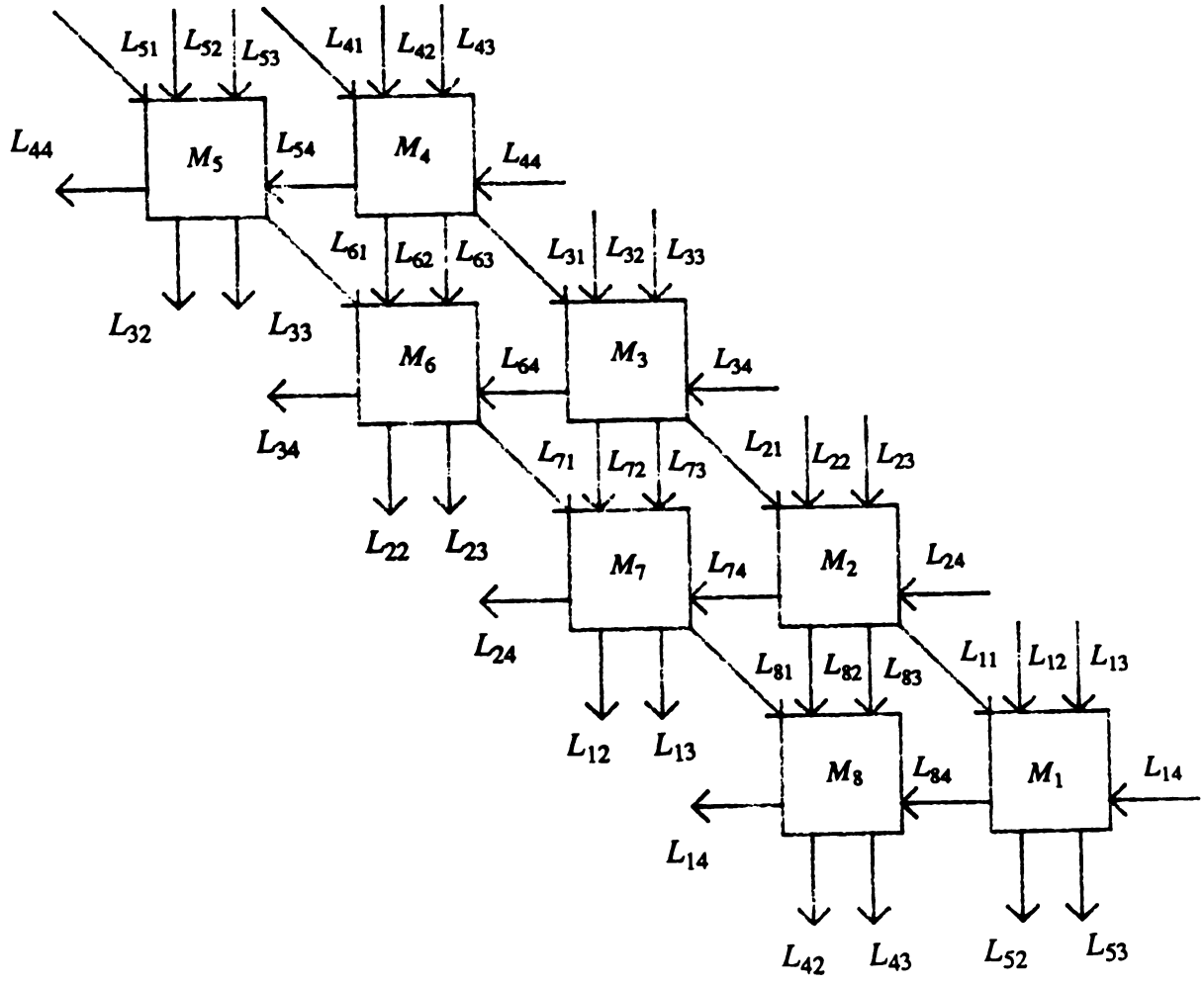


Figure 25. Eight Basic Cells of a MRSD.

Theorem 9.

The set of labels,

$$L_1=(L_{11},L_{12},L_{13},L_{14})=(V_1,\overline{C}_2,C_1\oplus K,V_2),$$

$$L_2=(L_{21},L_{22},L_{23},L_{24})=(V_1,\overline{V}_2,C_4\oplus K,V_3),$$

$$L_3=(L_{31},L_{32},L_{33},L_{34})=(V_1,\overline{C}_2,C_3\oplus K,V_2),$$

$$L_4=(L_{41},L_{42},L_{43},L_{44})=(V_1,\overline{V}_2,V_2\oplus K,V_3),$$

$$L_5=(L_{51},L_{52},L_{53},L_{54})=(C_1,\overline{C}_2,C_2\oplus K,C_4),$$

$$L_6=(L_{61},L_{62},L_{63},L_{64})=(C_1,\overline{V}_2,V_4\oplus K,V_1),$$

$$L_7=(L_{71},L_{72},L_{73},L_{74})=(C_1,\overline{C}_2,C_4\oplus K,C_4),$$

and $$L_8=(L_{81},L_{82},L_{83},L_{84})=(C_1,\overline{V}_2,C_2\oplus K,V_1)$$

is a solution of the mappings (22) with the cell function (21), where K is either 1 or 0.

Proof: Similar to the proof of Theorem 7, this can be simply proved by solving (22) under D=1.

Similarly consider the case of D=0, the mapping (20) can be simplified as

YXAZ ----> PSB, (23)

where P=f($\overline{X}$, Y, Z),

S=g(X, Y, Z),

and B=g(A, Y, Z).

The following Theorem can be concluded.

Theorem 10.

The set of labels,

$L_1=L_3=(V_1,\overline{C}_3,C_3\oplus K,V_4)$,

$L_2=L_4=(V_1,\overline{V}_2,V_2\oplus K,V_3)$,

$L_5=L_7=(C_1,\overline{C}_2,C_2\oplus K,C_4)$, and

$L_6=L_8=(C_1,\overline{V}_4,V_4\oplus K,C_3)$,

is a solution of the mapping (22) with the cell function (23), where K is either 1 or 0.

Proof : Similar to the proof of Theorem 7, this can be simply proved by solving mappings (22) under D=0.

In summary, Theorems 11 and 12 describe the labels applied to a MRSD for both D=1 and 0, respectively. Table 19 illustrates the input combinations of the MCS cells of Figure 25, where $L_i=(L_{i1},L_{i2},L_{i3},L_{i4})$. In Table 19, while Test #1-#8 and #11-#18 are the input combinations derived directly from Theorem 12 for D=0, Test #21-#28 and #31-#38 are those derived from Theorem 11 for D=1. Similar to Table 17, in order to apply all possible input combinations, Tests #9, #10, #19 #20, #29, #30, #39, and #40 are added.

Table 19. Input Combinations for MRSD.

Test#	D	L_1 (bafp)	L_2 (bafp)	L_3 (bafp)	L_4 (bafp)	L_5 (bafp)	L_6 (bafp)	L_7 (bafp)	L_8 (bafp)
1.	0	0100	0100	0100	0100	0100	0100	0100	0100
2.	0	0101	0101	0101	0101	1010	1010	1010	1010
3.	0	0011	0010	0011	0010	1100	1011	1100	1011
4.	0	0100	0011	0100	0011	1101	1100	1101	1100
5.	0	1011	1100	1011	1100	0010	0011	0010	0011
6.	0	1100	1101	1100	1101	0011	0100	0011	0100
7.	0	1010	1010	1010	1010	0101	0101	0101	0101
8.	0	1011	1011	1011	1011	1011	1011	1011	1011
9.	0	0010	0010	0010	0010	0010	0010	0010	0010
10.	0	1101	1101	1101	1101	1101	1101	1101	1101
11.	0	0110	0110	0110	0110	0110	0110	0110	0110
12.	0	0111	0111	0111	0111	1000	1000	1000	1000
13.	0	0001	0000	0001	0000	1110	1001	1110	1001
14.	0	0110	0001	0110	0001	1111	1110	1111	1110
15.	0	1001	1110	1001	1110	0000	0001	0000	0001
16.	0	1110	1111	1110	1111	0001	0110	0001	0110
17.	0	1000	1000	1000	1000	0111	0111	0111	0111
18.	0	1001	1001	1001	1001	1001	1001	1001	1001
19.	0	0000	0000	0000	0000	0000	0000	0000	0000
20.	0	1111	1111	1111	1111	1111	1111	1111	1111
21.	1	0100	0100	0100	0100	0100	0100	0100	0100
22.	1	0010	0101	0000	0101	1010	1110	1000	1110
23.	1	0111	0000	0111	0010	1100	1010	1100	1000
24.	1	1000	1100	1010	1100	0010	0111	0000	0111
25.	1	1101	1010	1111	1010	0101	0001	0111	0001
26.	1	1011	1011	1011	1011	1011	1011	1011	1011
27.	1	0000	0000	0000	0000	0000	0000	0000	0000
28.	1	1111	1111	1111	1111	1111	1111	1111	1111
29.	1	0011	0001	0011	0001	0001	0011	0001	0011
30.	1	1110	1100	1110	1100	1100	1110	1100	1110
31.	1	0110	0110	0110	0110	0110	0110	0110	0110
32.	1	0000	0111	0010	0111	1000	1100	1010	1100
33.	1	0101	0010	0101	0000	1110	1000	1110	1010
34.	1	1010	1110	1000	1110	0000	0101	0010	0101
35.	1	1111	1000	1101	1000	0111	0011	0101	0011
36.	1	1001	1001	1001	1001	1001	1001	1001	1001
37.	1	0010	0010	0010	0010	0010	0010	0010	0010
38.	1	1101	1101	1101	1101	1101	1101	1101	1101
39.	1	0001	0011	0001	0011	0011	0001	0011	0001
40.	1	1100	1110	1100	1110	1110	1100	1110	1100

4.2.2. Design for C-testability.

Figure 26 shows the Modified 4-by-4 Restoring Array Divider (MRSD). Similar to Figure 21, additional XOR gates and control lines, Test1 and Test2, are needed. During the normal operation, Test1, Test2 and z's are all assigned to logic 0, and terminals a's, b's and r_0-r_2 are discarded.

4.2.3. Test Pattern Generation.

According to Table 19, Algorithm 6 generates the test patterns and expected outputs for a MRSD. Table 20 illustrates the test patterns and expected outputs for a 4-by-4 MRSD. Therefore, the following Theorem results.

Theorem 11.

The MRSD of Figure 26 is C-testable with a test length of 40.

Proof: Similar to the proof of Theorem 2, if M_{BC} is defined as the eight basic cells of RSD and TS is the 40 test vectors in Table 19, then an n-by-n MRSD can be partitioned into pq M_{BC}'s, where n=2p=4q. Since each M_{BC} can be tested by the test set TS, an n-by-n MRSD can then be tested by the same test set TS. So, the MRSD is C-testable with a test length of 40.

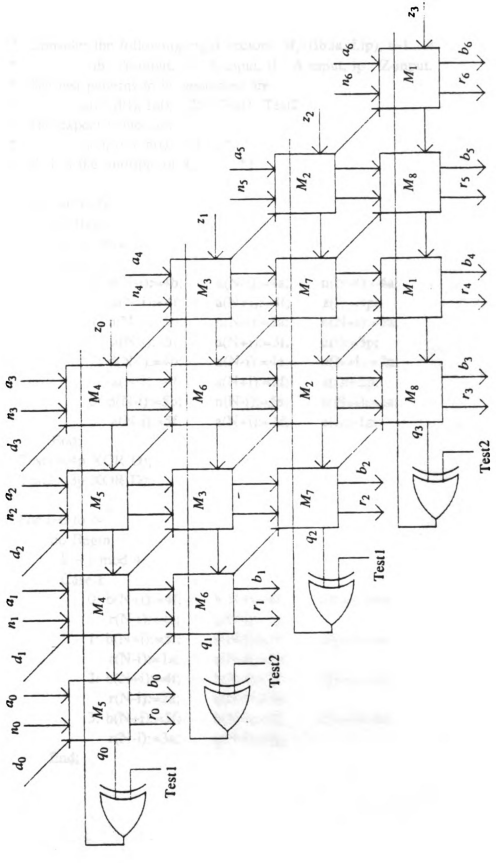


Figure 26. A Schematic Circuit Diagram of a 4-by-4 MRSD.

Algorithm 6 :

```
{* Consider the following eight vectors: M_i: (ib,ia,if,ip), i=1....4
 *          ib : Y-input, ia : X-input, if : A-input, ip : Z-input.
 * The test patterns to be generated are :
 *        a(i), d(i), i=0,...,2N, Test1, Test2
 * The expect values are :
 *        q(i), r(i), b(i), i=0,...2N
 * N+1 is the multiple of 4        *}

  For i=0 to N
      do Begin
       k = i mod 4
       Case k
           0: d(N-i):=4b;    n(N-i):=4a;    n(N+i):=4a;
              a(N-i):=4f;    a(N+i):=4f;    z(i):=4p;
           1: d(N-i):=5b;    n(N-i):=5a;    n(N+i):=3a;
              a(N-i):=5f;    a(N+i):=3f;    z(i):=3p;
           2: d(N-i):=4b;    n(N-i):=4a;    n(N+i):=2a;
              a(N-i):=4f;    a(N+i):=2f;    z(i):=2p;
           3: d(N-i):=5b;    n(N-i):=5a;    n(N+i):=1a;
              a(N-i):=5f;    a(N+i):=1f;    z(i):=1p;
      End;
 Test1=(4p XOR D);
 Test2=(3p XOR D);

 For i=0 to N
      do Begin
       k = i mod 4
       Case k
           0: b(N+i):=4f;    b(N-i):=4f;    r(N+i):=4a;
              r(N-i):=4a;    q(N-i):=1p;
           1: b(N+i):=5f;    b(N-i):=1f;    r(N+i):=5a;
              r(N-i):=1a;    q(N-i):=2p;
           2: b(N+i):=4f;    b(N-i):=2f;    r(N+i):=4a;
              r(N-i):=2a;    q(N-i):=3p;
           3: b(N+i):=5f;    b(N-i):=3f;    r(N+i):=5a;
              r(N-i):=3a;    q(N-i):=4p;
      End;
```

Table 20. Test Patterns and Expected Outputs of a 4-by-4 MRSD.

Test#	D	n	d	a	z	Test1	Test2	r	q	b
1.	0	1111111	0000	0000000	0000	0	0	1111111	0000	0000000
2.	0	0101111	1010	1010000	1111	1	1	1111010	1111	0000101
3.	0	1010000	1010	0101111	0101	0	1	0000101	0101	1111010
4.	0	1010101	1010	0101010	1010	1	0	1010101	1010	0101010
5.	0	0101010	0101	1010101	0101	0	1	0101010	0101	1010101
6.	0	0101111	0101	1010000	1010	1	0	1111010	1010	0000101
7.	0	1010000	0101	0101111	0000	0	0	0000101	0000	1111010
8.	0	0000000	1111	1111111	1111	1	1	0000000	1111	1111111
9.	0	0000000	0000	1111111	0000	0	0	0000000	0000	1111111
10.	0	1111111	1111	0000000	1111	1	1	1111111	1111	0000000
11.	0	1111111	0000	1111111	0000	0	0	1111111	0000	1111111
12.	0	0101111	1010	0101111	1111	1	1	1111010	1111	1111010
13.	0	1010000	1010	1010000	0101	0	1	0000101	0101	0000101
14.	0	1010101	1010	1010101	1010	1	0	1010101	1010	1010101
15.	0	0101010	0101	0101010	0101	0	1	0101010	0101	0101010
16.	0	0101111	0101	0101111	1010	1	0	1111010	1010	1111010
17.	0	1010000	0101	1010000	0000	0	0	0000101	0000	0000101
18.	0	0000000	1111	0000000	1111	1	1	0000000	1111	0000000
19.	0	0000000	0000	0000000	0000	0	0	0000000	0000	0000000
20.	0	1111111	1111	1111111	1111	1	1	1111111	1111	1111111
21.	1	1111111	0000	0000000	0000	1	1	1111111	0000	0000000
22.	1	0101010	1010	1010001	1010	0	1	0101010	1010	0010101
23.	1	1010101	1010	0101101	0101	1	0	1010101	0101	1011010
24.	1	0101010	0101	1010100	0000	1	1	0101010	0000	1000101
25.	1	1010101	0101	0101110	0101	1	0	1010101	0101	1101010
26.	1	0000000	1111	1111111	1111	0	0	0000000	1111	1111111
27.	1	0000000	0000	0000000	0000	1	1	0000000	0000	0000000
28.	1	1111111	1111	1111111	1111	0	0	1111111	1111	1111111
29.	1	0000000	0000	0000101	1111	0	0	0000000	1111	1010000
30.	1	1111111	1111	0000101	0000	1	1	1111111	0000	1010000
31.	1	1111111	0000	1111111	0000	1	1	1111111	0000	1111111
32.	1	0101010	1010	0101110	1010	0	1	0101010	1010	1101010
33.	1	1010101	1010	1010010	0101	1	0	1010101	0101	0100101
34.	1	0101010	0101	0101011	0000	1	1	0101010	0000	0111010
35.	1	1010101	0101	1010001	0101	1	0	1010101	0101	0010101
36.	1	0000000	1111	0000000	1111	0	0	0000000	1111	0000000
37.	1	0000000	0000	1111111	0000	1	1	0000000	0000	1111111
38.	1	1111111	1111	0000000	1111	0	0	1111111	1111	0000000
39.	1	0000000	0000	1111010	1111	0	0	0000000	1111	0101111
40.	1	1111111	1111	1111010	0000	1	1	1111111	0000	0101111

Remark : $n=(n_0\ n_1\ n_2\ n_3\ n_4\ n_5\ n_6)$ $d=(d_0\ d_1\ d_2\ d_3)$ $a=(a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6)$
$z=(z_0\ z_1\ z_2\ z_3)$ $r=(r_0\ r_1\ r_2\ r_3\ r_4\ r_5\ r_6)$ $q=(q_0\ q_1\ q_2\ q_3)$
$b=(b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6)$

V. Conclusions

Based on the fault model that a faulty cell can permanently change its truth table in any arbitary way as long as it remains a combinational circuit, the design of C-testable array multipliers and dividers are presented. The proposed design of C-testable Array Multipliers can be tested with 16 test patterns, and the designs of Non-restoring Array Divider and Restoring Array Divider are tested with 20 and 40 patterns, respectively. The schematic designs and the algorithm generating test patterns are also provided. The study shows that the test patterns and expected outputs can be systematically generated by using the graph labeling scheme. By using of graph labeling scheme, a set of test vectors with repetition nature can be obtained. This drastically simplify the test generation procedure.

It has been shown that the proposed design of Carry-Propagate Array Multiplier is better than that of [9] in the performance of speed because the XOR gates are removed from the critical path. The proposed design of Baugh-Wooley Array Multiplier is better than that of [8] in the number of test patterns required, i.e., it only requires 16 patterns of the proposed design, but 55 patterns in [8]. In addition, the proposed design can systematically generate the patterns by a circuit with limited size. Also, the proposed design of MRSD is C-testable, but that of [9] is not.

Although the C-testable design requires a test set of constant length irrespec-

tive of the circuit size, and the test cost can be significantly reduced, it is not free of penalty. Extra XOR gates and control signals are needed to produce the C-testability. In practice, the extra hardware for XOR gates may not be the burdern of the design. However, the most important and critical problem is the extra control signals. This study has shown that the number of extra control signals may be as many as triple that of the circuit size. It is impractical to increase the number of pins in the design of arithmetic units to produce C-testability because the number of pins of an IC package is limited.

From Algorithms 1-6, the test patterns are generated by a set of vectors that are derived from the labels of the basic four or eight cells. These test vectors can be generated internally and repetitively. This leads to an excellent platform for the developement of the BIST (Built-in Self Test) design of C-testable arithmetic units.

LIST OF REFERENCES

[1] Abraham, J.A., and V.K. Agarwal, "Test generation for digital systems," in Fault-tolerant Computing, Theory and Techniques, edited by D.K. Pradhan, Prentice-Hall, Englewood Cliffs, N.J., 1986.

[2] Lala, P.K., Fault Tolerant & Fault Testable Hardware Design, Prentice-Hall, 1985.

[3] Kautz, W.H., "Testing for faults in cellular logic arrays," Proc. 8th Symp. Switching Automata Theory, 1967, pp.161-174.

[4] Friedman, A.D., "Easily testable iterative systems," IEEE Trans. on Computers, Vol.C-22, pp.1061-1064, Dec. 1973.

[5] Parthasarathy, R., and S.M. Reddy, "A testable design of iterative logic arrays," IEEE Trans. on Circuits and Systems, Vol.CAS-28, pp.2037-1045. Nov. 1981.

[6] Hwang, K., Computer Arithmetic, Principles, Architecture, and Design. Wiley, New York, 1979.

[7] Waser, S., "High speed monolithic multipliers for real-time digital signal processing," Computer, pp.19-28, Oct. 1978.

[8] Shen, J.P., and F.J. Ferguson, "The design of easily testable array multipliers," IEEE Trans. on Computers, Vol.C-33, No.6, pp.554-560, June 1984.

[9] Chatterjee, A., and J.A. Abraham, "Test generation for arithmetic unit by graph labeling," Proc. of 17th Fault-Tolerant Computing Symp., Pittsburgh, PA, pp.284-289, July 1987.

[10] Patel, J.H. "Testing in two dimensional iterative logic arrays" Fault-Tolerent Computing, IEEE, 1986, pp. 76-81

[11] Elhuni, H. "C-testability of two-dimensional iterative array" IEEE Trans. on Computer-Aided Design, VOL CAD-5 No.4, pp.573-581, Oct. 1986