FACTPLA
FUNCTIONAL ANALYSIS AND THE COMPLEXITY OF TESTING PROGRAMMABLE LOGIC ARRAYS

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

By

S. I. ABBAS (BSc., MSc.)

DEPARTMENT OF ELECTRICAL & ELECTRONIC ENG.

BRUNEL (THE UNIVERSITY OF WEST LONDON)

UXBRIDGE, MIDDLESEX, U.K., UB8 3PH

MAY 1988

ABSTRACT

A computer aided method for analyzing the testability of Programmable Logic Arrays (PLAs) is described. The method, which is based on a functional verification approach, estimates the complexity of testing a PLA according to the amount of single undetectable faults in the array structure.

An analytic program (FACTPLA) is developed to predict the above complexity without analyzing the topology of the array as such. Thus, the method is technology invariant and depends only on the functionality of the PLA. The program quantitatively evaluates the effects of undetectable faults and produces some testability measures to manifest these effects.

A testability profile for different PLA examples is provided and a number of suggestions for further research to establish definitely the usefulness of some functional properties for testing were made.

TO MY PARENTS,
BROTHERS & SISTERS

Samir

ACKNOWLEDGMENT

I should like to express my sincere appreciation and deep gratitude to prof. G. Musgrave for his agreeing to be my supervisor and for his guidance, patience, and continuous encouragement throughout this study.

Thanks are due to Dr. T. Alukadi and Dr. M. Hadjinicolaou for their useful and stimulating discussion and to all my colleagues at Brunel 'CAD group'.

CONTENTS

CHAPTER ONE

INTRODUCTION

1.1 INTRODUCTION

The rapid evolution of semiconductor technology towards higher device densities has increased the effort to prove the design validation, manufacturing quality, and longer term operational reliability. With increasing circuit density, a vast amount of data processing and storage are required to perform testing. Examples of this type indicate that a test pattern generation program could run for several weeks for complex circuits like that of a Micro Vax computers. This can account for about 60% of the cost of test. It has been noticed that the best measure of the test effectiveness has involved fault simulation programs; the most costly part of test preparation in Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) environments. Such programs are usually based on a simple fault model for the circuit under test. The cost of the test may increase to such an extent as to regard the circuit as untestable, i.e., the cost of test pattern generation, evaluation and application are beyond the budget.

Most automated test generation procedures assume 'single' faults, since multiple faults analysis is, in general, much more complicated [1]. The reason for such complexity come from the fact that multiple faults assume some sort of signal independencies that may defeat fault sensitization required to generate the test patterns. This implies that every single fault is detected and repaired before some other fault can occur.

In practice, however, most digital circuits tend to contain various undetectable faults. Due to the phenomenon of masking among faults, the existence of such undetectable faults has a great influence on testing high scale integrated circuits.

In the following review the basic issues and problems in testing digital circuits are briefly discussed. Recent techniques to overcome the testing problem, namely design for testability and testability analysis, are also introduced.

1.2 FAILURE MECHANISMS

The design of a digital system can be viewed as a sequence of transformations of design representations at different levels of abstraction [2] :

(a) functional (informational) representation,
(b) logic (gate) representation,
(c) physical (circuit or geometric) representation.

Thus, when modeling a circuit malfunction, an appropriate model for the malfunction must be established at each level. This is a "failure" in the circuit level, "fault" in the gate level, and "error" in the informational level.

However, for ease of computation, the fault model should possess some basic properties :

1. easily definable and manipulatable by computer,
2. able to reflect a variety of technological failures, and

3. able to give a satisfactory coverage of the actual failures.

In digital environments, a switching variable can have one out of two logical values; ZERO and ONE. Hence, fault modeling is easier to handle at the gate representation and most fault detecting techniques have adopted this policy. Accordingly, the stuck or stuck_at fault model represents a typical logical fault model which is frequently considered. In this model, an arbitrary signal line k in the circuit is assumed to be fixed permanently at either logical 0, k stuck at ZERO (k S@ 0), or logical 1, k stuck at ONE (k S@ 1).

For testing purposes, single stuck at fault model, in which the circuit is assumed to contain at most one stuck fault, is widely used. More complex gate_level fault models are the multiple stuck and bridging faults. In the multiple stuck fault model, it is assumed that signal lines can have values that are fixed and independent of the other signals in the circuits. The bridging fault model assumes that two signal lines are connected together so that a wired logic occurs.

However, the above classes of gate_level faults have some drawbacks [3] :

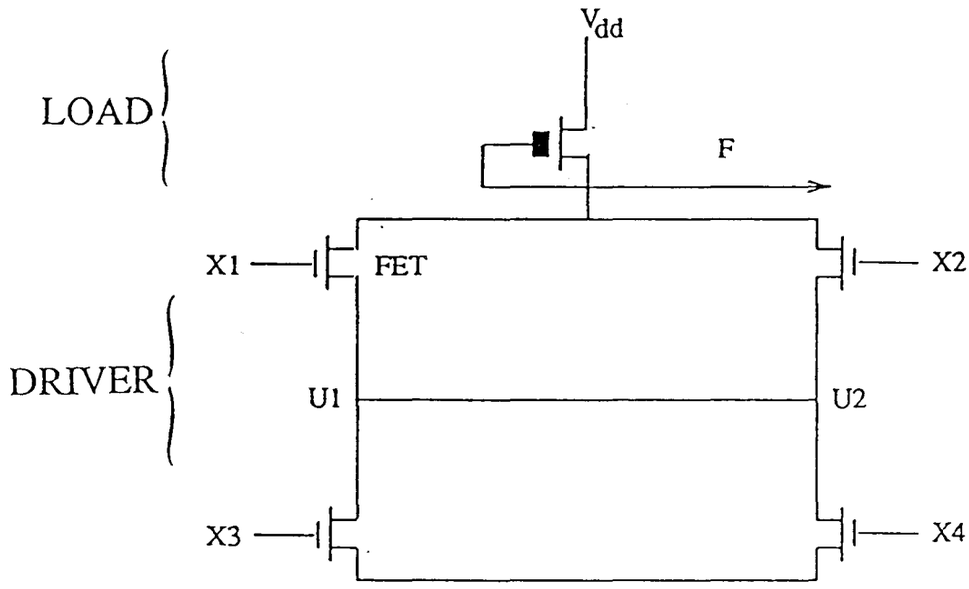
1. there are many switch_level faults which do not fall into these classes. Some switch_level shorts and opens can not be modeled by many gate_level faults because they involve a modification of the function realized by the relevant logic diagram.

2. the logic diagram does not always constitute a real model of the circuit diagram. For instance, in MOS technology many stuck_at faults in a gate_level representation of a MOS device do not correspond to any physical failures of interest.

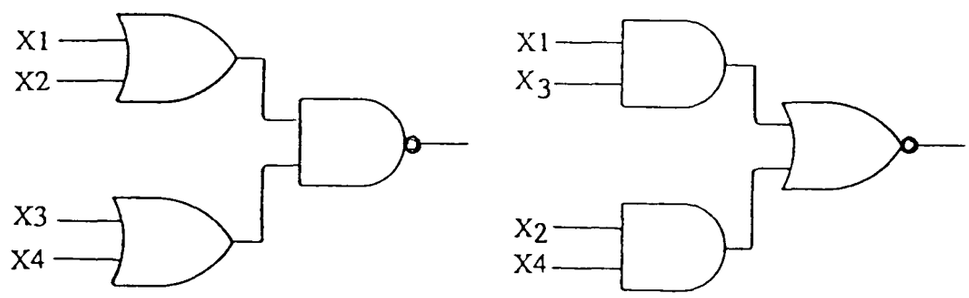
3. with increasing circuit density, logic_level representation results in large number of component blocks. Also, when considering bi_directional switches then as many as 15 logic levels have to be considered. The complexity of testing and the number of test patterns will increase accordingly.

Figure 1.1 illustrates a failure that can not be modeled as a stuck fault. The nMOS complex gate given in Figure 1.1 consists of a LOAD and a DRIVER. If the input to a Field Effect Transistor (FET) is logic 1, then a conducting path between its other leads exists. Otherwise, these leads do not conduct. If, due to some input assignment, a conducting path between the output of the gate and ground exists, then the output of the gate is at logic ZERO. However, if there is no such path, then the depletion transistor in the LOAD section of the gate will pull up the output to V_{dd} (logic ONE). Obviously, there are 4 possible conducting paths which may force the output to logic ZERO.

Now, if the permanent connection between U1 and U2 in the above gate is broken for some reason, then only two conducting paths remain, i.e., $F = \overline{X_1X_3 + X_2X_4}$. This 'faulty' function does not correspond to any stuck fault



(a) a Complex nMOS Gate



(b) original function (c) modified function

Fig. 1.1 Effect of Circuit _level Faults

in the equivalent logic diagram of the gate.

Accordingly, fault models, other than stuck and bridging faults, have gained popularity. Current efforts are being directed towards modeling failures at both switch and functional levels. An advantage of testing switch_level failures is that in some cases it may be possible to utilize the structural properties (regularity, for instance) of the circuit to obtain a much simpler test set compared to the one generated to the gate_level representation. However, testing switch_level failures involves a much greater amount of complexity. On the other hand, functional_level fault models help reduce the complexity of testing highly dense circuits. In this case, a very accurate fault model is required so that it can reflect the interconnection structure of the transistor circuit.

1.3 TESTING PROBLEMS

In today´s technology, the Integrated Circuit (IC) chip represents the smallest physical element which has to be considered in the manufacturing environments. At the end of the IC fabrication process, the wafer disk (approximately 75mm in diameter) is containing many copies (called die) of the IC arranged in a rectangular matrix. Typically, only a minority, perhaps 30%, of these copies work properly. Hence, the actual step of testing begins at the fabricated wafer [4]. Figure 1.2 shows a simplified flow chart for testing the ICs. Test T1 is performed to

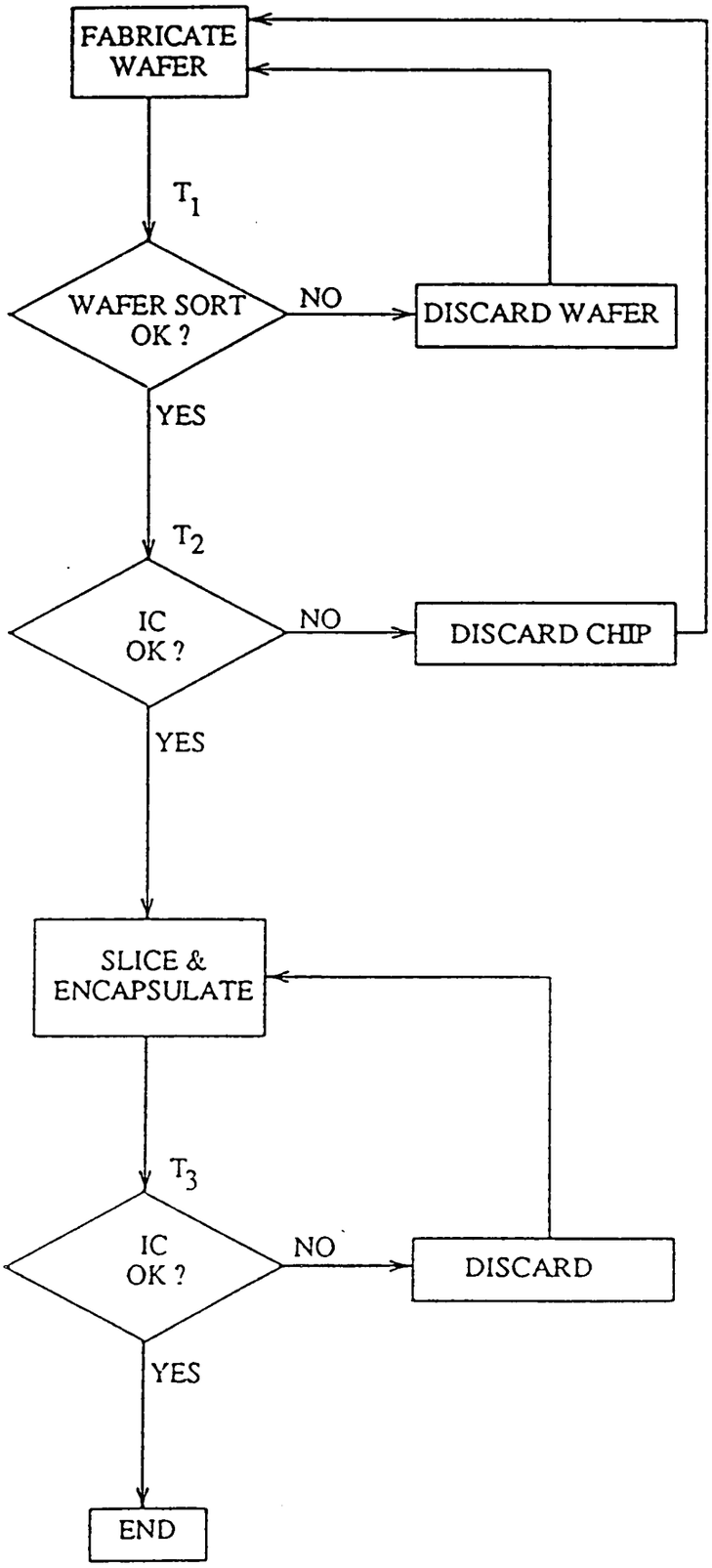


Fig. 1.2 IC Manufacturing & Testing

check whether the fabrication process has been carried out satisfactory.

Having established that the wafer contains an acceptable percentage of good circuits, the actual copies must now be tested (test T2) on wafer. Three different types of IC testing can be recognized :

1. wafer sort : each IC on the wafer must be tested to identify the good circuits,
2. parametric testing : parameters such as propagation delay and drive currents are checked, and
3. functional testing : determine whether or not the IC carries out the function for which it was designed.

Due to access limitations on wafer, parametric and functional testing are restricted at this stage. Hence, the IC must also be tested after it is packaged (test T3), after it is mounted on a board, and perhaps periodically after it is placed in a system. However, among these, functional testing has been shown to be the most expensive part of the process. As the technology progresses towards high levels of integration, the ratio between the number of devices on a chip to the chip input/output pins increases considerably. Such chip ´complexity´ has caused a rapid increase in the cost of performing functional testing.

In the following, the problems involved in testing digital circuits are presented in terms of two parameters; the cost of functional testing and the effects of undetectable

faults.

1.3.1 Cost of Functional Testing

The basic procedure of functionally testing digital circuits involves three main activities [5] :

1. Test Pattern Generation (TPG) : deriving and selecting a set of input stimuli which is either 'exhaustive', 'random', or 'algorithmically' generated.

2. Test Pattern Evaluation (TPE) : justifying the effectiveness of the test patterns using one of two different techniques; fault simulation and fault insertion.

3. Test Application and Fault Finding (TAFF) : applying the test pattern to a real circuit by means of sophisticated Automatic Test Equipment (ATE).

Accordingly, the cost of testing digital circuits may be divided into :

1. TPG cost : relates to one of two parameters
   (i) the computer time required to run the TPG program plus the capital cost of developing the program (automatic),
   (ii) the number of man hours required for a person to write the test pattern plus the increase in system development time caused by the time taken to develop the tests (manual).

2. TPE cost : depends mainly on fault simulation techniques. Such techniques require an accurate model

of the circuit under test and repeated evaluation of circuit signals for each fault taken from the fault list. It has been proved that fault simulation run times represent the most considerable contributor to overall testing cost [5].

3. TAFF cost : depends on

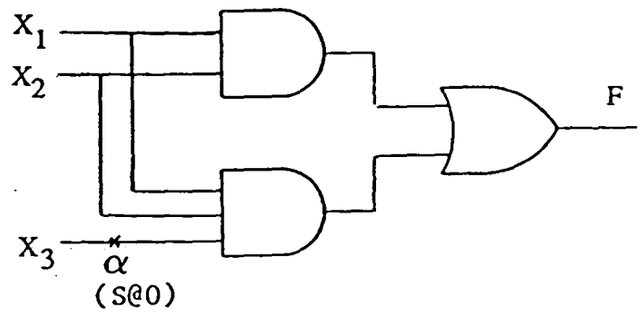
(i) the cost of Automatic Test Equipments and their interface requirements for different types of circuits,

(ii) the tester time required to apply the test.

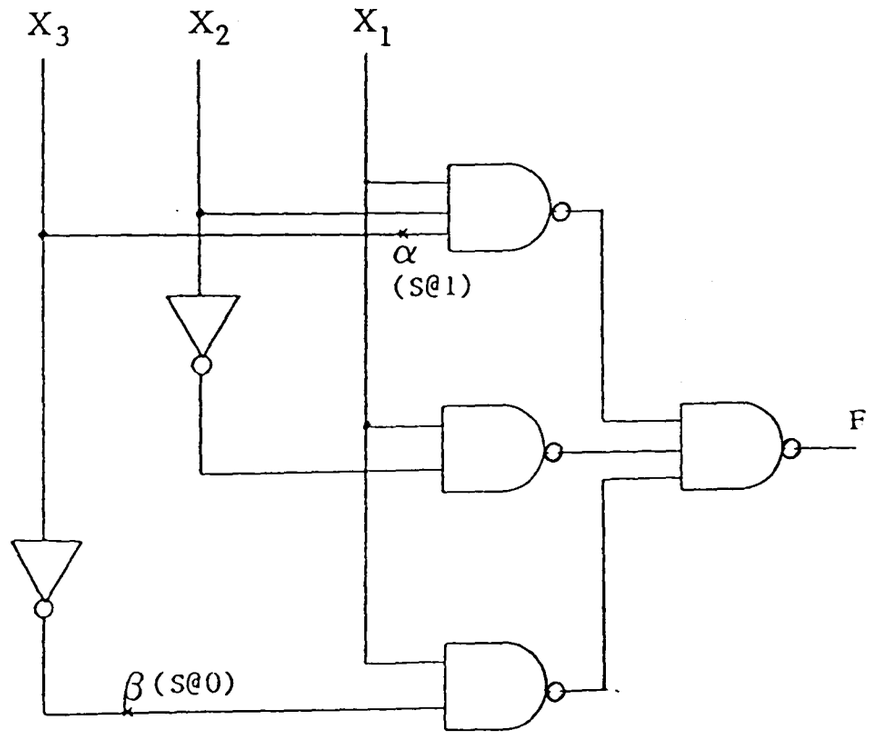
The Automatic Test Equipments have become very expensive and the computing time required to calculate the input patterns to be applied to the circuit has become very costly.

1.3.2 The effect of undetectable faults

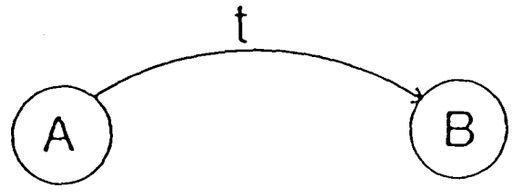
A digital circuit is said to be redundant with respect to some fault if the fault is undetectable, i.e., the function realized by the circuit with the fault present is equal to the function realized by the circuit without the fault [1]. In sketch (a) of Figure 1.3, the fault α S@0 (or S@1) is not testable since the output F is equal to logic 'ONE' if and only if $X_1=X_2=1$ regardless of whether this fault is present or not. Most testing methods are based on the assumption that the network under test is irredundant. Therefore, a great amount of computation may be wasted in trying to find tests for undetectable (or redundant) faults. Beside, the existence of undetectable faults in any circuit has a great influence on two of the



(a) Undetectale fault α



(b) Fault α Masks Fault β



(c) Direct Graph Representation

Fig. 1.3 Illustrations of undetectable Faults in Digital Circuits

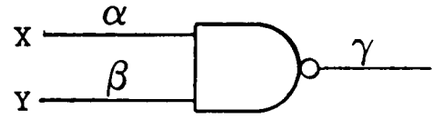
most important aspects in functional testing :

(i) Fault Masking

In a general digital circuit, a non_detectable fault may mask the detection of a normally detectable fault for a given test pattern. An example of fault masking is shown in sketch (b) of Figure 1.3. The fault α (S@1) is undetectable at the output node F because of the inconsistent assignment at inputs X_1, X_2, and X_3. The fault β (S@0) is detectable at F by the input test T_β (110), which is the only test available for this fault. However, the presence of α (S@1) prevents (masks) the detection of β (S@0). The above masking phenomenon is defined below :

Definition 1.3.1. A redundant fault f_i masks a testable fault f_j under the test T if and only if T detects f_i but does not detect the combined fault $f_i f_j$.

Fault masking phenomenon may also be described by a direct graph (see sketch c of Figure 1.3). Generally, each node in the graph represents a fault (single or multiple) and an arc goes from node A to node B with label t if fault A masks faults B under test t, i.e., B is detected but AB is not. If the 'masking graph' is designed for a given test set, i.e., a complete single fault test set, the labels of its arcs may be omitted [29]. Note that fault masking may occur in irredundant circuits as well. As an illustration, consider the following NAND gate :



The test "X=0, Y=1" detects the single fault "line α S@1". If α S@1 and γ S@1 are both present, then the above test will not detect the fault α S@1. This masking is not a problem as the test "X=1, Y=1" detects the fault γ S@1.

Nevertheless, in a general combinational circuit, the existence of loops of successively masking faults is greatly affects the ability of single fault test sets to cover multiple faults as well.

(ii) Multiple Fault Detection

In a general digital circuit, there are many more multiple faults than single faults. A circuit with k lines may contain up to $(3^k - 1)$ possible multiple faults [1]. Accordingly, the consideration of various multiple faults would be extremely impractical. on the other hand, the single fault model does not accurately cover various failures which can occur in highly dense circuits. However, previous research on multiple faults has come to the assumption that a complete single fault test set also detects most of the multiple faults [1]. The validity of this assumption is greatly affected by the amount of undetectable faults and the possibility of fault masking occurrence.

1.4 TESTABLE DESIGN ISSUES

Undoubtly, the discussion given in the last section has indicated that it is not enough just to design a circuit, after it is built it has to be tested easily. Furthermore, it has been justified that testing costs

contribute considerably (50% - 60%) in overall product cost. Current trends for increasing circuit density have significantly decreased system hardware cost. The implication of such trends had led to a situation in which:

1. testing is becoming more difficult since the number of pins per IC is not increasing in proportion to the number of gates per IC. Thus, the ability to 'control' and 'observe' signal values for the logic on chip is reduced, and
2. the percentage of system cost due to testing is increasing drastically.

It appears, therefore, that the only economical method to reduce testing costs is to include circuitry on each chip to facilitate testing. The expenditure, in the design stage, of adding such circuit overheads will result in overall reduction in cost. Two key concepts are involved in all strategies of designing testable circuits; controllability (a measure of the ability to set and reset every node internal to the circuit), and observability (a measure of the ability to observe the state of any node in the circuit). These circuit attributes reflect the degree of circuit testability. However, there exist two main techniques which relate to circuit testability :

1. Design For Testability (DFT) : addresses the testing problem during design by building testability in the circuit by design, and

2. Testability Analysis : a DFT method that approximates the difficulty of testing before generating the test patterns.

1.4.1 Design for testability

These techniques improve control and observation properties of the digital circuits. They may be divided into two main categories :

1. Ad Hoc : directed towards correcting specific design features that create testing problems. The most direct way to do this is to introduce test points, that is, additional circuit inputs and outputs to be used in the testing mode. These points allow the internal signals to be controlled and observed during testing. Such technique is seen to be specific and not generally applicable to VLSI and WSI due to the IC pin limitations. On the other hand, it has been estimated that the complexity of test pattern generating and fault simulation is proportional to the number of logic gates to the third power [7]. Therefore, the partitioning of large circuits and the individual testing of the sub_circuits should reduce the overall testing cost. However, this method is still unhelpful if the sub_circuits contain many memory devices which require initialization.

2. Structured : applying a set of design rules which are generally applicable. Basically, these techniques have been introduced to control and observe the

memory devices, therefore, the test generation problem could be reduced to the one of just testing the combinational logic. Scan techniques (Scan Path, Scan Set, Random Access Scan, and Level Sensitive Scan Design LSSD) permit access to internal nodes of a circuit without requiring a separate external connection for each node accessed. Scan Path is the most recognizable approach to describe the concept of these techniques. In the testing mode of Scan Path circuits, all registers or flip flops are converted into shift registers and connected as scan path for shifting in test patterns and shifting out test responses to a tester. The IBM LSSD uses this approach. However, due to the nature of serial data transmission this approach involves a time delay for applying the tests and receiving the responses. Furthermore, expensive test equipments and storage for a large number of test patterns and responses are required. Alternatively, Built in Self Testing (In_Chip Tester or Built_in Logic Block Observation) uses some silicon area to eliminate the disadvantages associated with Scan techniques. On the other hand, this method suffers many drawbacks. For instance, the extra silicon area devoted for the test circuitry reduces reliability and computational capacity of the chip, as well as efficient automated test procedure is required. A review of most DFT techniques is well presented in a paper by Williams and Parker [7] .

1.4.2 Testability Analysis

Having established the philosophy that additional overheads and general design constraints are very significant in the design method, it is vital that the designer has all the necessary information to make his design judgments. For instance, it is desirable to limit the "extra additions" to those necessary to insure adequate testability of the design. Therefore, a need for a 'testability measure' has become widespread. It is easy to argue that the use of an Automatic Test Pattern Generation (ATPG) program may provide a measure for the circuit testability. For example, parameters such as the run time of the program, the number of test patterns, and the fault coverage may constitute the above measure. However, two difficulties are associated with such direct approach :

1. the large expense involved in running the ATPG program, and
2. the lack of information about how to improve the testability of a circuit with poor testability measure.

Alternatively, testability analysis programs have been developed to estimate the design testability by predicting the cost (running time) of generating the test patterns. These programs tend to quantify the circuit properties of controllability and observability prior to determine its testability without actually running any ATPG program. TMEAS (Testability Measure Program [8]), SCOAP (Sandia

Controllability Analysis Program [9]), TESTSCREEN [10]. VICTOR (VLSI Identifier of Controllability, Testability, Observability, and Redundancy [11]), and CAMELOT (Computer Aided Measure for Logic Testability [12]) are the most popular testability measure programs.

1.5 SUMMARY

In the light of the above discussion, the very large scale digital circuits make testing extremely difficult, and when test patterns can not be obtained within the allowed computation time, it will be difficult to achieve a high rate of test coverage. Furthermore, multiple fault coverage must also be adequate since these faults represent an important parameter in VLSI environment. Testability analysis programs have been offered as solutions to the problems of test complexity and coverage. The information obtained by such programs may be used to develop some design constraints for achieving easily testable designs. For instance, advance knowledge of all redundant faults and their masking influences could be used to estimate the complexity of testing, multiple fault coverage capability of a single fault test set and, hence, the degree of testability for a given digital circuit. Clearly, the philosophy behind any test strategy may serve to systematically obtain such knowledge. Functional testing of 2_level AND_OR networks seems to be very attractive in this context.

The above concept represents the theme of the work established in this thesis as applied to programmable logic arrays; one of the most popular structures in VLSI design. FACTPLA (Functional Analysis and the Complexity of Testing PLAs) is a fast, compact and systematic program which has been developed to identify redundant faults and quantify their masking effects in general PLAs. The contact fault model has been adopted as it represents the most dominant failure model in these arrays. The program also indicates the way with which more realistic test patterns may be chosen for a PLA, i.e., which patterns should be included in the test set in order to cover as many multiple faults as possible.

Chapter 2 of this thesis introduces PLAs, as an increasingly powerful tool for LSI and VLSI design. Various types of faults that are apt to occur in a PLA are analyzed and the contact fault model is justified. A brief review of the previous strategies for testing PLAs is also discussed.

The theoretical strategy for the FACTPLA program is presented in Chapter 3. The method of prime implicant testing of irredundant 2_level AND_OR networks, as suggested by Kohavi [13], is elaborated prior to analyze the effects of undetectable faults in a PLA. This is analyzed and based upon the functional specification of a PLA (as a set of product terms), therefore generating testability criteria for the array structure. A simple hueristic is used by adopting the decimal code of the

product term as an ordered set of minterms. The policy of choosing the realistic tests for detectable faults is also justified.

Chapter 4 presents the implemented FACTPLA algorithms for predicting the impact of redundant contact faults on difficulty of testing the PLA. The applications and complexity of such algorithms are also discussed.

Suggestions to direct a future research towards deriving a more testable PLA structure and generalizing the applications of the FACTPLA program are summarized in Chapter 5.

CHAPTER TWO

AULT ANALYSIS IN PROGRAMMABLE LOGIC ARRAYS

2.1 INTRODUCTION

Considerable technological achievement over the last few years has produced an increasing level of integration of components onto a single Integrated Circuit (IC) chip. Very Large Scale Integration (VLSI) plays a major role in the development of complex digital systems.

Standard ICs have been produced to meet several requirements of both simple combinational circuits and the more complex sequential networks. There are some applications, however, that require a circuit which is not available in standard IC form. The concept of custom or semi_custom design has been introduced to provide flexibility and easy design changes. In practice, different digital semi_custom ICs can be classified as belonging to one of the following main groups [2]

(i) Gate Arrays (or Uncommitted Logic Arrays, ULAs)
(ii) Cell_Based Systems
(iii) Programmable Logic Devices (Matrix Logic)

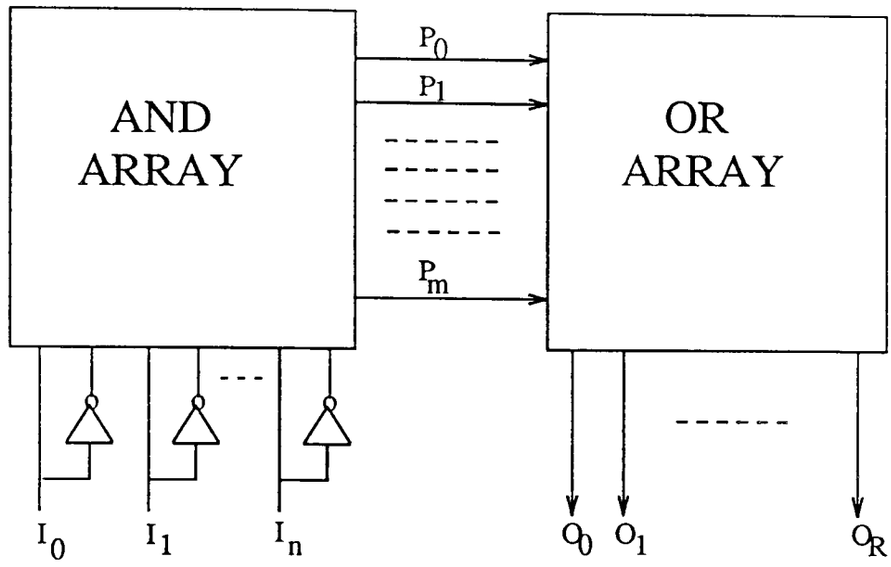
Programmable Logic Devices are basically capable of implementing arbitrarily complex logic functions in sum of products (SOP) forms. Depending on the manufacturer, the best_known examples of these devices are the familiar PROM (Programmable Read Only Memory) and FPLA (Field Programmable Logic Array). Recently, PAL (Programmable Array Logic), FPGA (Field Programmable Gate Array) and FPLS (Field Programmable Logic Sequencer) have been introduced and added to the category of these devices.

Figure 2.1 represents an overall architecture which summarizes the comparison between PROM, FPLA and PAL. It is obvious that all these devices perform similar logic functions, the differences are determined by whether the AND matrix, the OR matrix or both are programmable.

2.2 THE STRUCTURE OF PROGRAMMABLE LOGIC ARRAYS

Basically, the PLA is a logical element designed to produce sum of products expressions. It allows more efficient use of the silicon area by representing a set of logic functions in a compressed yet regular form. A PLA implements a two stage combinational logic through a pair of adjacent rectangular arrays. Thus, conceptually a PLA may be viewed as a collection of 2_level AND_OR or NAND_NAND or NOR_NOR networks.

The PLA is an important building block for VLSI circuits. It is commonly used to design instruction decoders of microprocessors, and combinational circuitry of finite_state machines. The key technological advantage of using a PLA in an IC technology relies on the straightforward mapping between the symbolic representation and its physical implementation. Any combinational logic function can be described by a logic cover. For PLAs this logic cover is represented by a pair of matrices, called input and output matrices. The symbolic _ physical mapping is obvious in Figure 2.2. In this Figure, each input variable of the logic function is represented by a column in the input matrix in sketch (a).



(a) PROM : AND array pre_programmable. Fully decoded n inputs yield to all $m = 2^n$ possible product terms.

OR array user_programmable. Any combination of the m products can be ORed onto one of the R outputs.

(b) FPLA : AND array user_programmable. Generate a chosen products of m ($<2^n$) products from n inputs.

OR array user_programmable. Any combination of the m products ($<2^n$) can be ORed onto one of the R outputs.

(c) PAL : AND array user_programmable. Generate a chosen products of m ($<2^n$) products from n inputs.

OR array pre_prograamble. Groups of products are ORed onto the R outputs according to a pre_arranged pattern.

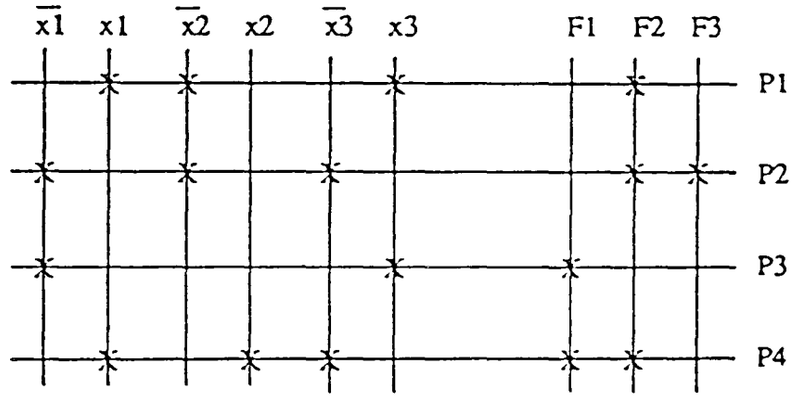
Fig. 2.1 a Comparison of Programmable Logic Devices

In sketch (b) of Figure 2.2, each variable corresponds to a pair of columns in the left part of the physical implementation. Every implicant of the logic function is represented by a row in the symbolic representation . The input part of each row represents a logical product of some input variables. Finally, every output of the logic function corresponds to a column in the right part of the physical array.

The implementation of a particular logic function is performed by "programming" the PLA, i.e., by placing (or connecting) an appropriate device at the intersection points (between the rows and columns) specified by '1' or '0' in the symbolic representation. The grid (intersection) points between the input/output columns and the product lines (rows) are called crosspoints. The left part of the PLA structure shown in Figure 2.2(b) is called the SEARCH or PRODUCT TERM array, while the right part is called the READ or SUMMING array. A cross (contact) in the SEARCH array represents the presence of an input variable (uncomplemented or complemented) in the relevant product term, while a cross in the READ array represents the presence of a product term in some output function. Accordingly, every PLA will have specific personalities (devices or fusible links) which define the overall structure personality.

Input Matrix	Output Matrix
1 0 1	0 1 0
0 0 0	0 1 1
0 * 1	1 0 0
1 1 0	1 1 0

(a) A Simple PLA (Symbolic)



(b) the PLA Representation

Fig. 2.2 Symbolic_ Physical Representation in PLAs

PLAs are sized or specified by the following description

n * m * p

where

n : number of input variables,

m : total number of unique p_terms,

p : number of output functions.

Thus, in a general (n,m,p)_PLA structure, there are m(2n+p) crosspoints.

Partitioning the inputs to a PLA is performed with the aid of a Bit Partitioning Network (BPN). Single and Double decoders are the most common BPN used in PLAs [6].

2.2.1 PLA Implementation

PLAs are compatible with different technologies. For example, in the AND_OR bipolar transistor implementation, the AND matrix is implemented with diodes, and the OR matrix is implemented with bipolar transistors. This provides the designer with the ability to program his array by blowing fusible links within the array.

In another fabrication method, the implementation is possible with MOSFET technology where the presence or absence of gate connections define the function realized. This is the case with NMOS technology where NAND gates can be made as well as NOR gates. However, in MOS technology, it is convenient to exploit the use of NOR gates rather than NAND gates (in terms of performance and ease of design). Thus, the NOR_NOR NMOS PLA implementation is most common in VLSI MOS circuits [2].

However, a simple schematic diagram (identical to the physical representation) has been adopted to represent the PLA implementation of switching functions. For instance, the bipolar FPLA realization and the MOSFET implementation of the PLA shown in Figure 2.2 is illustrated in Figure 2.3 and 2.4 respectively.

For fault analysis and test generation purposes, the personality of a PLA has been defined using two different procedures for input and output columns [14]. In other words, the functionality (personality) of a PLA can be represented by a 0_1 matrix defined as follows :

definition 2.2.1 given a row and an "output" column in a fault free PLA, if there is a cross between the two (or, there is no cross), then a 1_contact (correspondingly, a 0_contact) is said to exist between the row and the column.

definition 2.2.2 given a row and an "input" column in a fault free PLA, if there is a cross between the two (or, there is no cross), then a 0_contact (correspondingly, a 1_contact) is said to exist between the row and the column.

The personalized crosspoints between input lines and rows will be referred to as input contacts, while the personalized crosspoints between output lines and rows will be called output contacts.

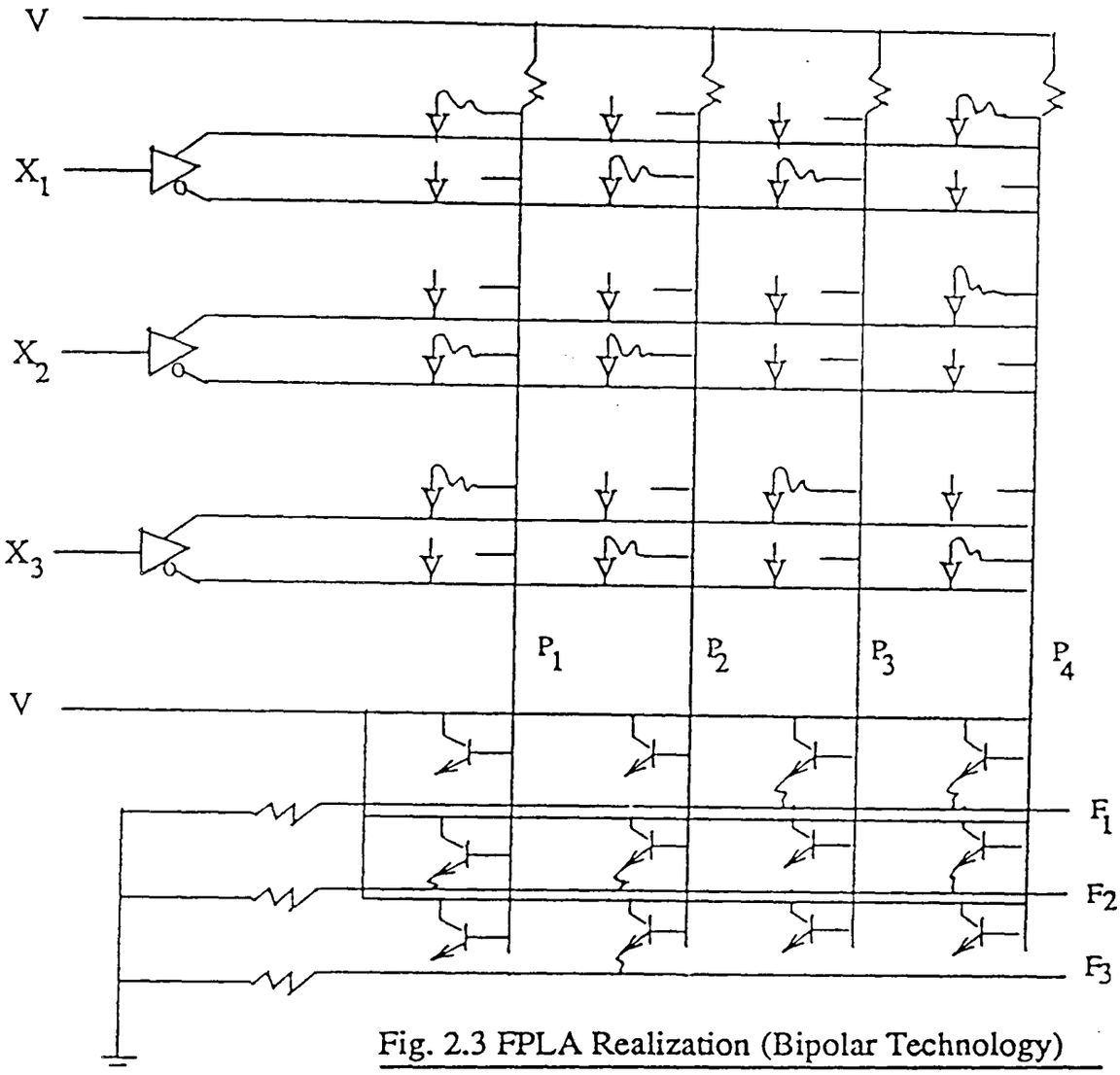


Fig. 2.3 FPLA Realization (Bipolar Technology)

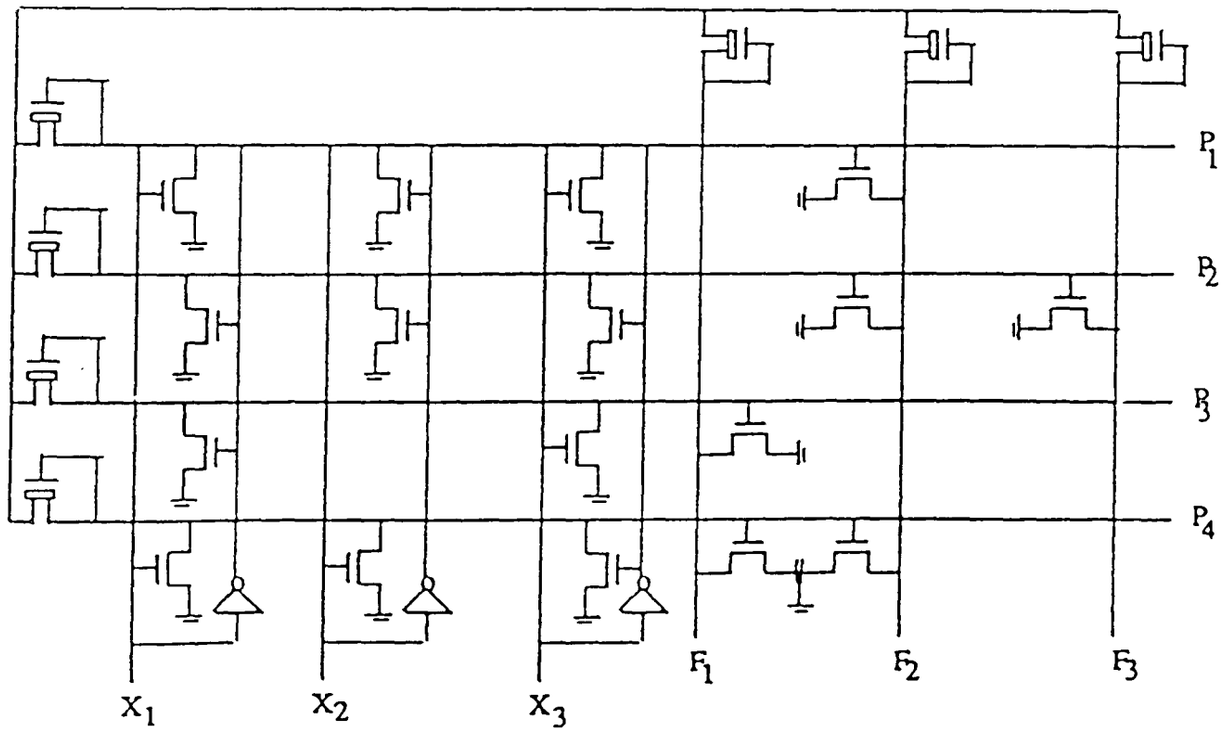


Fig. 2.4 nMOS NOR_NOR PLA (MOSFET Technology)

The personality matrix for the PLA structure shown in Figure 2.2 is given below :

$\bar{X}_1$	X_1	$\bar{X}_2$	X_2	$\bar{X}_3$	X_3	F_1	F_2	F_3
1	0	0	1	1	0	0	1	0
0	1	0	1	0	1	0	1	1
0	1	1	1	1	0	1	0	0
0	1	1	0	0	1	1	1	0

2.2.2 PLA Folding

Being general purpose and programmable, the PLA is not always a dense layout for specific functions. Most PLA personality matrices are very sparse, so that a straightforward mapping to physical design will result in a significant waste of the silicon area. It is possible to recover some of this lost area by topological manipulation of the array. This kind of manipulation is known as "folding", but it has the disadvantage of reducing the permutability of the inputs and outputs at the same time as reducing the area [2].

Row and column folding of a PLA are techniques which attempt to reduce the area by exploiting its sparsity. Figure 2.5 shows a 5_input, 6_output, 8_product terms PLA using an 'equivalent' area for a 5_input, 4_output, 4_product terms PLA plus six pull_ups. It is easy to realize that the OR plane is folded on both sides of the AND plane which is situated in the middle. All rows and some columns have been split to allow for more product

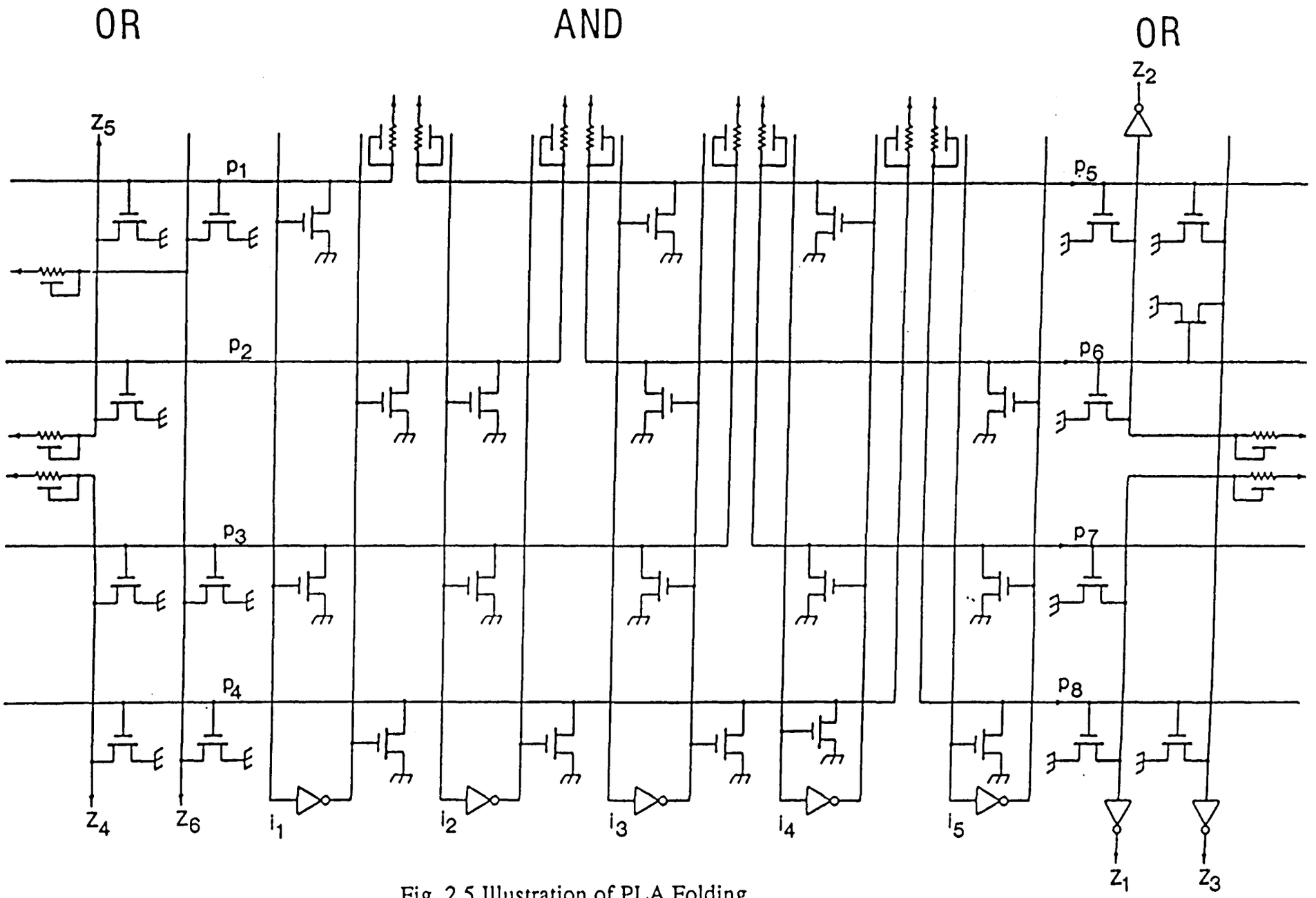


Fig. 2.5 Illustration of PLA Folding

terms and output functions, thereby resulting in a denser PLA. It should be noticed, however, that the realization of such structure imposes certain decomposability properties on the output functions. For example, Z_2 does not depend on inputs i_1 and i_2; Z_5 is independent of i_3, i_4, i_5 and so on. Therefore, the design of compact PLA structures requires very complex decomposability and functional separability algorithms.

2.2.3 Impact of PLAs on Logic design

In additional to cost effectiveness and optimum memory features, PLAs introduce some other design advantages :

1. Fast and smaller system design.

   The design and implementation time can be reduced considerably due to the programmability and flexibility of the arrays.

2. Easy design changes - edit flexibility.

   The uncomplemented or complement values of any input variable may be selected by some programming method. However, if both values are programmed (selected) in a product term, this term will never be selected since $X.\bar{X} = 0$. In FPLAs, the above feature may be used to deactivate (remove) any previously programmed product term. Moreover, unprogrammed inputs represent don't care assignment, therefore additional input variables can be added to the old product term at any time by programming the desired don't care conditions. Also, a sum of products expression can be expanded by adding new product terms. This can be

done by further a programming process in the OR plane of the PLA.

2.3 FAULT MODELING IN PROGRAMMABLE LOGIC ARRAYS

When testing digital circuits, most of the physical failures are modeled at different description levels using various types of fault models. For the analysis of faults in PLAs, the simple schematic diagram shown in Figure 2.2, is used. In that Figure, stuck at faults, shorts, and crosspoint defects are the most likely fault behavior to occur. In the rest of this section, the relationships among these faults are presented prior to justify the validity of crosspoint defects as the basic fault model for PLAs.

2.3.1 Stuck at Faults

With this fault model, one of the lines in the PLA is stuck permanently to one of the two logic values; it is commonly caused by a short to ground or to power.

Depending on the type of the implementational circuitry, the effect is considered as one or more lines of the equivalent logical model being stuck at 0 (S@0) or stuck at 1 (S@1). For example, for a PLA constructed with NOR technology, the value on a product line (or on an output line) is a NOR function of all the devices (contacts) on it. Thus, all stuck at faults in a general PLA structure can be represented by three fault categories in the equivalent 2_level NOR_NOR network. These fault categories are described below :

(a) Stuck at faults on input (bit) lines.

One of the input leads of the NOR gates in the first level is stuck at logic ZERO or logic ONE. The number of inputs to each NOR gate in this level is determined from the personality of the relevant product line in the search array. For fault detection purposes, only input lines S@0 of the NOR gate have to be considered [1].

In Figure 2.2, if an input line L is stuck at ZERO with only one contact on it, then a literal corresponding to this contact must have belonged to some product line P. Accordingly, P may be activated by a proper input stimuli and, hence, the effect of the S@0 fault will be equivalent to the missing contact between lines L and P. Therefore, the fault can be sensitized and the effect is propagated through one of the outputs containing P.

On the other hand, if there exist more than one contact on L, then L S@0 fault will be equivalent to a multiple contact fault whose components are all single missing contact faults on the same input column. These fault components are equivalent to single S@0 faults occurring simultaneously at the same logic level. It is well known that stuck at faults can not mask each other if they occur at the same logic level [16]. Therefore, the above fault is still detectable. However, if L has no contacts on it, then no sensitive path exists, and the fault is redundant.

(b) Stuck at Faults on product (word) lines.

These faults are considered as stuck at faults on the input leads of the NOR gates in the second logic level.

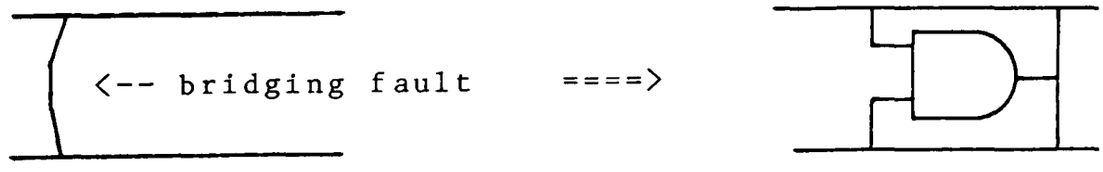
It is obvious from the previous discussion that a fault of this type is equivalent to some crosspoint defect where a contact at the crosspoint is missing.

(c) Stuck at Faults on output (function) lines.

In this case, both S@0 and S@1 faults must be considered. The faults are equivalent to missing and extra contact faults along the faulty output line. However, in a single output PLA, the output line has contacts on all its crosspoints with the product lines. Thus, all tests for the missing of these contacts cause the output line to have logical value 0. Accordingly, an output line S@0 fault is not guaranteed to be detected by such tests.

2.3.2 Bridging Faults

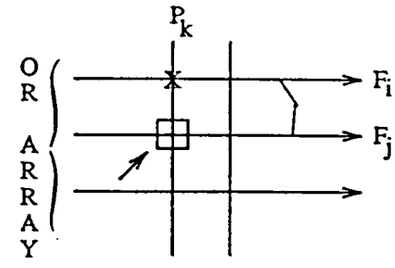
This is a short between two adjacent or crossing lines of the PLA. The commonly used stuck at fault fails to model logic circuit shorts [15]. Bridging faults are defined to model these circuit malfunctions. When two neighboring lines are connected accidentally, a wired logic is performed at the connection. Since there are two types of wired logic functions, namely the wired logic AND and the wired logic OR, therefore, there are two types of Bridging faults. For instance, in NMOS technology, an AND function between the shorted lines occurs :



Since most logic circuits are built by one of the logic families, only one type of bridging fault, either AND or OR will be considered at a time. However, if the shorted lines have the same personality then the resultant fault effect is undetectable since both lines assume the same logic value. Nevertheless, such fault has no influences on the functional operation of the circuit and, hence, the short is not important. Therefore, the key factor for investigating the effects of shorts is to assume different logic values on the affected lines. The set of all possible shorts in a PLA structure may includes the following :

(a) shorts between output lines.

The short is testable if there exist a device at the crosspoint between one of the shorted lines and one of the product lines, say P_k, such that no device between P_k and the other shorted line :

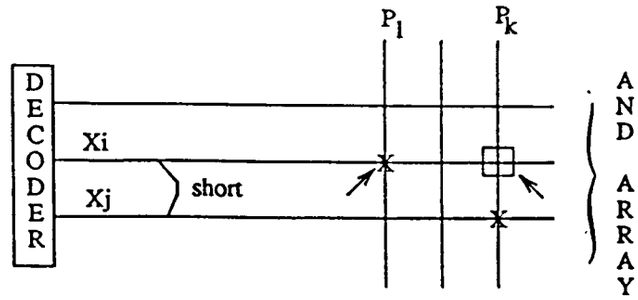


Any test pattern detecting the extra contact defect at the crosspoint between P_k and F_j is qualified to detect the short under consideration. Depending on the implementational circuitry, the short is detected on output F_i (ORed short) or on output F_j (ANDed short).

(b) shorts between input lines (one or two decoders).

Let X_i and X_j be the two shorted lines, then

with a single decoder : only one of the input lines has logic value ONE (decoder output) :

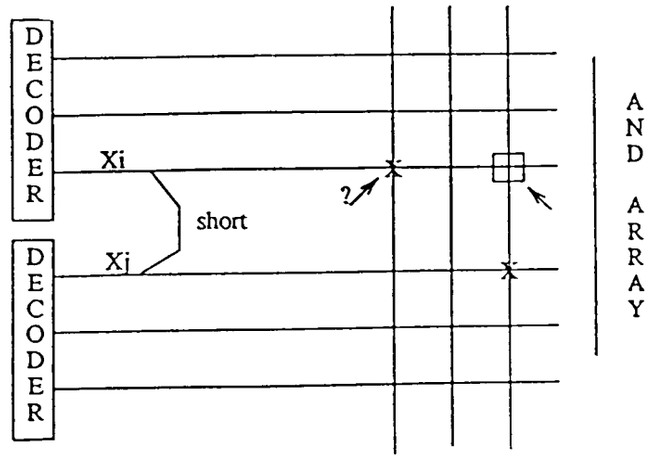


If $X_i = 1$, then two possible cases may be considered :

(i) X_j becomes logical ONE (ORed short). Hence, the short is detected by a test pattern for the extra contact defect at the crosspoint between X_i and product line P_k, such that X_j has a contact with P_k.

(ii) X_i becomes logical ZERO (ANDed short). Hence, the short is detected by a test pattern for the missing contact defect at the crosspoint between X_i and product line P_l, such that X_j has no contact with P_l.

with two decoders : consider the following case :

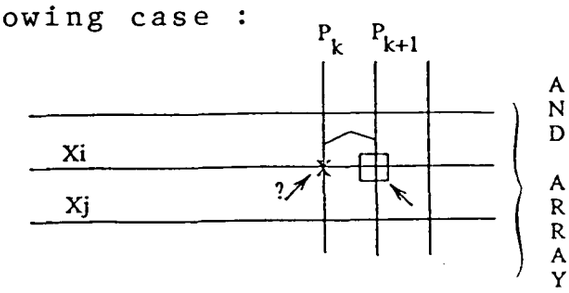


(i) if the short is ORed, the fault is detected in the same manner described in case (i) of the single decoder.

(ii) if the short is ANDed, the fault is detected in the same manner described in case (ii) of the single decoder if and only if $X_j = 0$. However, if $X_j = 1$, a test pattern other than those belonging to the contact defects test set is needed.

(c) shorts between product lines (AND plane).

Consider the following case :

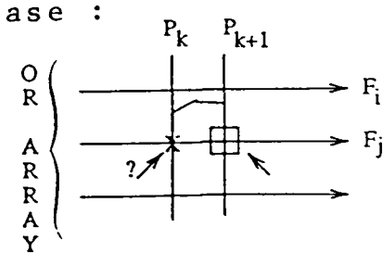


(i) if the short is ANDed, the fault is detected by a test pattern for the extra contact defect at the crosspoint between X_i and P_{k+1}.

(i) if the short is ORed, the fault is detected by a test pattern for the missing contact defect at the crosspoint between X_i and P_k provided that $P_{k+1} = 1$.

(d) shorts between product lines (OR plane).

Consider the following case :



(i) if the short is ORed, the fault is detected by a test pattern for the extra contact defect at the crosspoint between P_{k+1} and F_j.

(i) if the short is ANDed, the fault is detected by a test pattern for the missing contact defect at the crosspoint between P_k and F_j if and only if $P_{k+1} = 0$.

(e) crossline shorts (AND or OR planes).

In this case, the short occurs between the metalization and diffusion layers of the chip. An extensive analysis in [16] and [17] shows that a crossline short is equivalent to a multiple stuck fault at some logic level of the logical diagram. Therefore, these faults are guaranteed to be detected by some crosspoint defect tests.

Undoubtly, the above analysis reveals the influences of layout and personality of a PLA on the coverage of stuck_at and short fault. However, it has been concluded that any complete single crosspoint fault test set for a PLA is also a very good test for most stuck and bridging faults [16].

2.3.3 CROSSPOINT DEFECTS

This fault is the absence (missing) or the unnecessary presence (extra) of a cross connection or device between a bit line (input column) and a product line or between a product line and a sum (output) line. Recall the definitions of crosspoint contacts given in section 2.2.1, crosspoint defects may also be defined as follows :

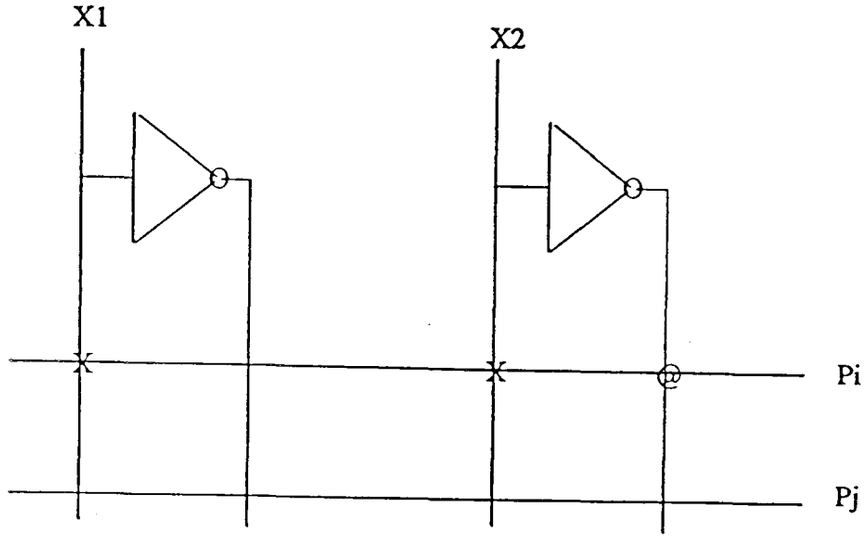
definition 2.3.1 a single 0_contact (1-contact) fault is said to exist in a PLA structure if due to some failure, a 0-contact (1_contact) of the fault free PLA becomes a 1_contact (0_contact) in the faulty PLA.

This type of fault is usually assumed for PLAs because it is more accurate than the other two types. The justifications for using the contact fault model may be summarized below :

1. fault collapsing can be performed within the contact fault model [18]. Extra contact fault, at the crosspoint of an already selected input variable in the AND plane, dominates a missing contact fault in the OR plane for the same row. For example, in NOR_NOR technology, the relevant product line is forced to zero as can be seen in Figure 2.6; showing such an arrangement and its equivalent representation. Thus, a test for this particular fault is qualified as a test for the "existence" of the product term P in the map of the output function.

2. the number of (single) crosspoint faults and crossline bridging faults is a function of the area of the PLA, while the number of the other (single) faults is linear in the number of input, output, and product lines. Since crossline bridging faults are equivalent to some stuck at faults (see previous section), then the number of the contact faults is by far the largest of all the three types. Hence, the size (length) of a single contact fault test set may also be the largest. Accordingly, a higher fault cover will be achieved by adopting this model.

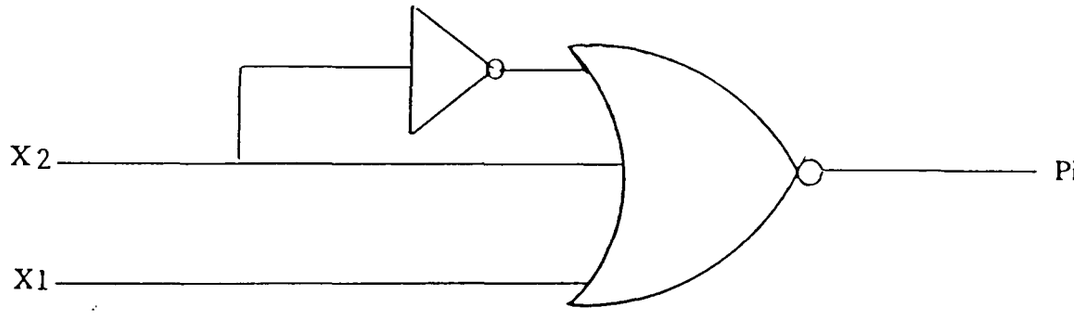
3. the contact fault model allows efficient generation of compact, technology invariant tests. The



X : normal contact

@ : extra contact (fault)

(a) modified personality of a faulty PLA



(b) equivalent representation in NOR_NOR technology

Fig. 2.6 Fault Collapsing in PLAs

structural regularity, and the ability of representing contact faults at a higher, functional level are good justifications.

2.3.4 THE PRODUCT TERM FAULT MODEL

As physical failures, crosspoint defects in a PLA may be viewed as incorrect wire connections on the equivalent logical diagram of the PLA. These connections are, in fact, the programming points (crosspoints) of the search and read arrays of the PLA. Figure 2.7(a) shows a simple schematic diagram of a 4 input PLA implementing two switching functions :

$$F_0 = X_0X_1 + X_1X_2 + X_2X_3$$

$$F_1 = X_0X_1 + X_1X_2 + \bar{X}_1X_2\bar{X}_3$$

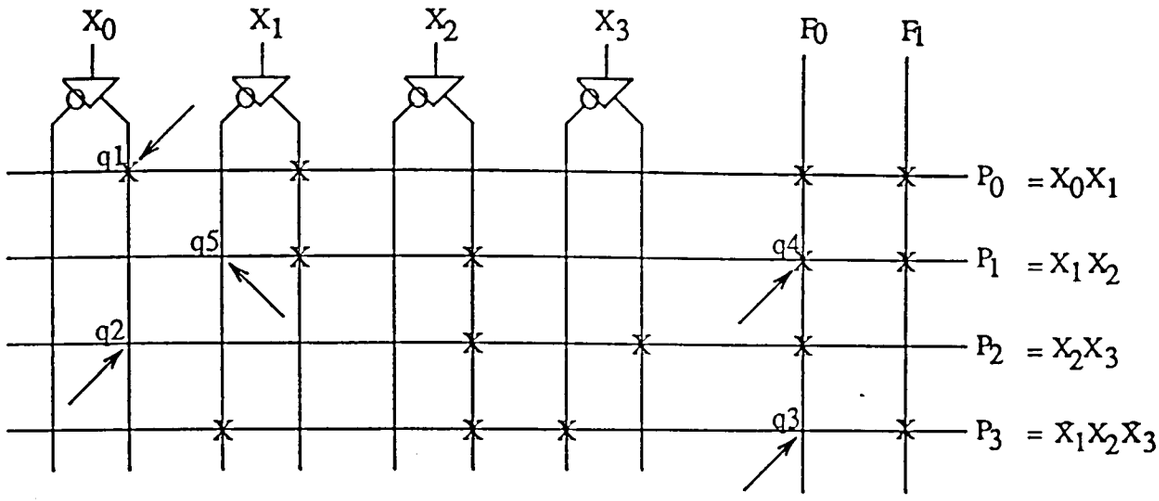
With the aid of Figure 2.7, the product term fault model is described below :

(1) Growth Faults, G

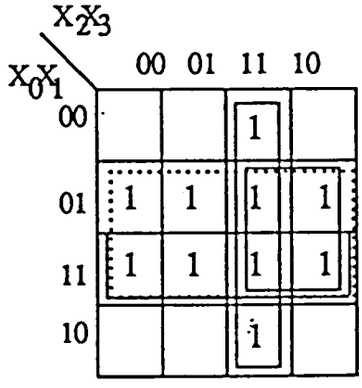
If an input literal is disconnected from an AND gate, the implicant generated by this gate will "grow" since it becomes independent of some input variable. The effect is equivalent to a missing contact fault in the search array. In Figure 2.7(a), a missing contact fault (q1) in the search array causes the implicant P_0 (X_0X_1) to grow to (X_1) as is illustrated in Figure 2.7(b).

(2) Shrinkage Faults, S

If an input literal becomes incorrectly connected to an AND gate, then the corresponding implicant "shrinks".

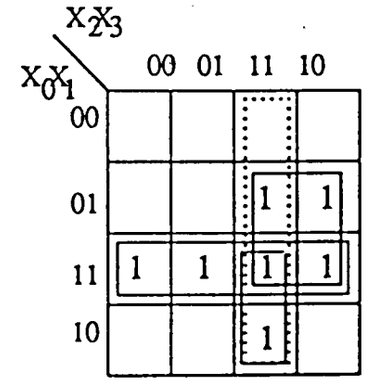


(a) Schematic Representation of a PLA

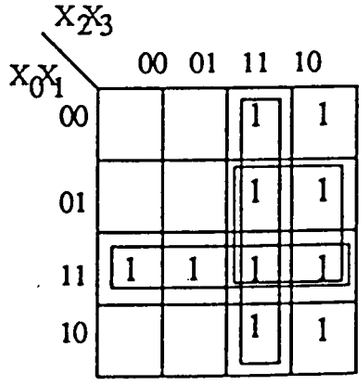


(b) Fault q1 : Growth of P_0

F_0

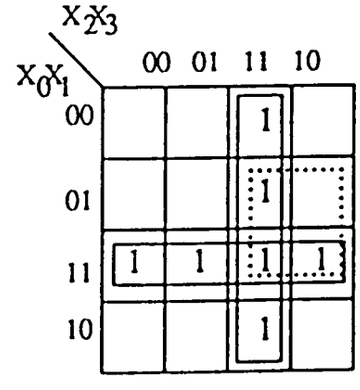


(c) Fault q2 : Shrinkage of P_2



(d) Fault q3 : Appearance of P_3

F_0



(e) Fault q4 or q5 : Disappearance of P_1

Fig. 2.7 Effects of Contact Faults on PLA Product Terms

For example, an extra contact fault (q2) causes the implicant P_2 (X_2X_3) to shrink to $(X_0X_2X_3)$ as shown in Figure 2.7(c).

(3) Appearance Faults, A

If an AND gate becomes incorrectly connected to an OR gate, then an implicant "appears" on the map of the corresponding output. This is equivalent to an extra contact fault in the read array. The effect of the extra contact fault (q3) is shown in Figure 2.7(d)

(4) Disappearance Faults, D

If an AND gate becomes incorrectly disconnected from an OR gate, then the corresponding implicant "disappears" from the map of the relevant output function. Thus, a missing contact fault (q4) will cause the disappearance of the implicant P_1 as shown in Figure 2.7(e).

(5) Vanish Faults, V

If X_j $(\bar{X}_j)$ is an input variable already connected to some AND gate, then an incorrect connection of $\bar{X}_j$ (X_j) to the same AND gate will cause the corresponding implicant to "vanish". The effect is identical to the disappearance of an implicant from the map of the output function.

To summarize, therefore

1. an input 0_contact fault is the same as a growth fault,
2. an output 0_contact fault is equivalent to an appearance fault,

3. an input 1_contact fault is equivalent to either a vanish or shrinkage fault, and

4. an output 1_contact fault is the same as a disappearance fault.

2.4 TESTING PROGRAMMABLE LOGIC ARRAYS

The increasing popularity, in terms of structural regularity and flexible means of implementing logic circuits, of Programmable Logic Arrays (PLAs) has imposed the necessity to establish efficient test procedures for these arrays.

Several approaches have been reported to generate a minimum or near minimum test set for a PLA. They all are affected, to some extent, by the basic fault model they use and by the size of the PLA. In the following subsections, a brief discussion on the use of various fault models to generate tests for PLAs is presented.

2.4.1 Using The PLA Logic Model

Muelhdrof [19] and Cha [20,21] have used classical stuck at fault test generation algorithms after modeling the PLA as a functionally equivalent logic network. For example, in MOSFET Technology, NOR gates are usually used to implement the required function. Accordingly, it is possible to employ a program that will use a stuck at zero (S@0) criterion for all input lines of all NOR gates, and both S@0 and S@1 faults for all output lines of all NOR gates to generate a complete test set. The procedure always involves the selection (activation) of one product

always involves the selection (activation) of one product line by assigning suitable input values (activity pattern contains 0, 1, and X : do not care) on the input columns of the PLA.

It is obvious that the computational time of the above test scheme increases considerably with the size of the PLA due to the large number of component blocks produced by adopting such a scheme.

2.4.2 Using The PLA Personality

Ostapko [22] used an abstract matrix representing the AND_OR personality of the PLA. In this method, for every single crosspoint defect, it is necessary to determine the equivalent bit pattern change that results from that defect. Thus, given a PLA personality, crosspoint fault detection is the same as testing that the ZEROS and ONES of the personality matrix are functionally correct. Each row in the personality matrix is regarded as a multi_part cube where the number of the cube parts depends on the number of decoder networks. The method uses global cube ordering and cube operations to derive the tests. The resulting bit change can be analyzed to see whether or not it would be detected. Thus, during the test generation process the method requires repeated fault simulation to determine the fault cover.

Eichelberger [23] associated with every used crosspoint defect a stuck at fault and established the necessary and sufficient conditions to sensitize a test path through the OR plane of the PLA. The method uses a matrix representing

the PLA without expansion into equivalent logic blocks. It exploits the concept of redundant testing and expands its application to generate tests for PLAs.

As the complexity of the PLA increases, random testing becomes inefficient due to the large number of 'used' crosspoints. The probability of detecting a missing crosspoint with a random pattern is not better than $1/2^n$, where n is the number of used crosspoints [23].

2.4.3 Using The PLA Functional Specification

Somenzi [18], Smith [17], and Bose [14] have used the cubical notation (see Appendix A) to represent the PLA personality and to specify the set of the product terms. Figure 2.8(a) shows a PLA specification matrix of the following switching function :

$$F_1 = X_1X_3\bar{X}_4 + X_2\bar{X}_3 + \bar{X}_1X_3$$

$$F_2 = X_1X_3\bar{X}_4 + X_2\bar{X}_3 + \bar{X}_2X_3X_4$$

In [18], fault simulation is required to establish an Excitation Cube, EC, representing the fault effect. The EC is obtained by assigning proper logic values at the faulty bit of the cube under consideration (see Figure 2.8(b)). Fault effect propagation is performed by selecting those conditions in EC which cause one of the outputs to depend on the fault to be covered. The necessary condition for preventing fault propagation is defined as a Masking Function MF; which is obtained always by deleting the faulty cube from the PLA specification matrix as shown in Figure 2.8(c).

A complete test set for a particular crosspoint defect is given by

EC - MF

where (-) denotes the "set difference" operation, i.e., to obtain those conditions in EC not covered by MF (see Appendix A).

The methods described in [14] and [17] use the same policy given in [18] as they analyze a crosspoint defect at a higher level. The effect of actual physical failures is viewed in terms of changes in the product term configuration on a Karnaugh Map, that is growth, shrinkage, appearance, and disappearance of the product terms.

In [17], fault simulation and backtrack procedures are required to perform fault sensitization to the outputs and consistency assignments to the inputs. On the other hand, the method described in [14] involves mass computations to perform the necessary comparisons between the cubes representing the product terms. The complexity of the algorithms employed by this approach grows geometrically with the number of the product terms in the PLA.

2.5 FAULT MASKING IN PROGRAMMABLE LOGIC ARRAYS

An important problem in fault detection is to verify whether a single fault test set is able to detect all multiple faults. A test derived for the detection of some fault may fail this purpose in the presence of another

fault [29]. Similarly, a set of diagnostic tests derived for a general PLA structure is not necessary a valid set, if a fault occurrence in the structure is preceded by the occurrence of some undetectable (redundant) faults. If a testable fault is masked, the output may indicate no fault during testing yet give erroneous response during normal operation. The above phenomenon, called "masking" among faults, has a great impact on fault detection in PLAs.

definition 2.5.2. A single contact fault (q_1) is said to be masked by another single contact fault (q_2) for an input vector X_t, if X_t tests q_1 but does not test the simultaneous fault q_1q_2.

Referring to the logical view of the contact fault model, the following properties specify all the necessary conditions under which masking might take place in a PLA.

In these properties let

G : denotes the set of all growth faults,

A : denotes the set of all appearance faults,

S : denotes the set of all shrinkage faults,

V : denotes the set of all vanish faults,

D : denotes the set of all disappearance faults, and

$\bigcup$: denotes the "set union" operation (see appendix A)

property 2.5.1 For a given test vector X_t, a detectable fault from the set $(G \bigcup A)$ existing on some product line L_i of the PLA, can be masked only by 'one' fault from the set $(S \bigcup D \bigcup V)$ also existing on the same product line L_i.

property 2.5.2 For a given test vector X_t, a detectable fault from the set $(S \bigcup D \bigcup V)$ existing on some product line L_i of the PLA, can be masked only by a fault from the set $(G \bigcup A)$ existing on some product line other than L_i.

The above two properties are based on lemmas 2 and 3 respectively as defined by Agarwal [25] where equivalent relationships have been proved.

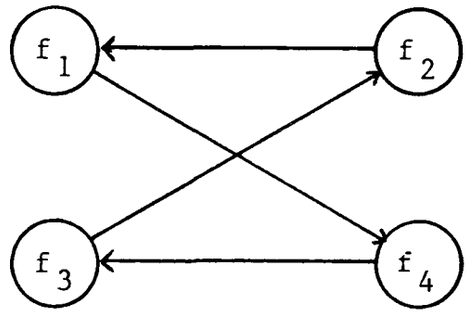
2.6 MULTIPLE FAULT DETECTION IN PLAs

The problem of multiple fault detection in PLAs is mostly directed to the evaluation of the single fault test set capabilities to detect multiple faults. The evaluation is based on a rather general assumption ; an irredundant PLA structure.

Agarwal [25] has proposed a modeled network, called Stuck At Equivalent, SAE, network, which represents the complete PLA structure. Each single crosspoint defect (contact fault) is functionally equivalent to some stuck at fault in the modeled network. The SAE network is shown to be a 3_level, internal fanout free with respect to any output line, network. In an irredundant, internal fanout free network, every multiple stuck at fault of size 2 or 3 (number of simultaneous faults) is covered by any test set that covers only the single stuck at faults of the network [26,27]. Accordingly, it has been stated that :

"Every complete single contact fault test set of an irredundant PLA covers every multiple fault of size 2 or 3" [25].

For multiple contact faults of size 4 and larger, the concept of the 4-way masking cycle is involved. This phenomenon occurs when four components (f_1, f_2, f_3, and f_4) of a multiple fault of size 4 or more, are distributed such that f_1 masks f_4, f_4 mask f_3, f_3 masks f_2, and f_2 masks f_1. This phenomenon may be described by the following direct graph :



An irredundant PLA with n inputs, m product terms, and p outputs has been proved to have the following property :

"Out of $\binom{m(2n+p)}{r}$, $r \geqslant 4$, different contact faults of size r, at most $\binom{m}{2}.(n + p/2)^4.\binom{m(2n+p)-4}{r-4}$ faults are not covered by every complete single contact fault test set of the PLA".

At the worst case, the maximum number of contact faults of size 4 with 4-way masking is $\binom{m}{2}.(n + p/2)^4$, as reported in [25].

A close investigation to the above coverage Figures shows that for an irredundant PLA with n=16, m=48, and p=8, there are two cases to be considered :

<u>case (i) r=4 :</u> the number $\binom{m}{2}.(n + p/2)^4$ is a mere .03% of the total $\binom{m(2n+p)}{4}$.

Thus, 99.97% of all multiple contact faults of size 4 are to be covered by each single contact test set.

case (ii) r>4 : in this case, if r becomes large the bound $\binom{m}{2}.(n + p/2)^4.\binom{m(2n+p)-4}{r-4}$ becomes greater than $\binom{m(2n+p)}{r}$. Thus, a practical use of the above property is only convenient for values of r not exceeding 8.

Similar evaluations have been described in [28] and [29] by Rajski and Tyzer. They quantitatively predicted the multiple contact fault coverage capability of a single contact fault test set in a PLA. The problem is studied from the point of view of the theory of combinations. They have shown that some of the multiple faults of size r, $r \geqslant 4$, which contain a 4-way masking could be detected by a single fault test set of the PLA. The validity of this point depends on the types and locations of the fault components other than those involved in the 4-way masking cycle.

2.7 SUMMARY

Test sets derived for the detection of single crosspoint defects in a PLA can not be safely used, if the PLA contains undetectable crosspoint faults. This is due to the phenomenon of masking among faults. The necessary conditions under which fault masking might take place in PLAs are presented using the product term fault model. The work presented in [25,28,29] is based on an "irredundant" PLA. The attempt was to predict the ability of single fault test sets to detect multiple faults. The

coverage results given in the above papers, however, can not be used for PLAs having redundant faults, unless the PLA is converted to a crosspoint irredundant structure for testing purposes.

No known method exists to convert a general PLA structure to an irredundant one without using extra hardware. For example, the control input procedure described by Ramanatha [30] implies that a number of control inputs (extra input columns) may be added to obtain a crosspoint irredundant PLA structure.

Other techniques have been proposed to augment a PLA for improving its testability [31-37]. They all are based on the idea of adding extra hardware to achieve high fault coverage and to overcome the problems of undetectable faults. Therefore, an extra silicon area must be devoted to serve for testing purposes, and faults in this additional test circuitry (mostly sequential) must also be considered.

In the following Chapter, the influences of undetectable contact faults in PLAs are analyzed. The prime implicant method for testing irredundant two level AND_OR networks [16] is elaborated for the sake of such analysis.

CHAPTER THREE

FAULT MASKING EVALUATION IN PROGRAMMABLE LOGIC ARRAYS

3.1 INTRODUCTION

The existence of undetectable faults represents one of the most important aspects in functional testing. Within the test generation process, even an exhaustive search may fail to find a test for a fault, i.e., no test exists. The fault is undetectable (or redundant) and the effort has been wasted. Therefore, efficient test generation requires advance knowledge of all redundant faults. In this context, the complexity of testing a digital circuit may be considered as related to the following parameters :

(a) The total number of redundant faults. This number has a great influence on the computational time of any automated test procedure.

(b) The multiple fault coverage capabilities of the single fault test set.

Redundant faults may be determined by analyzing the functional characteristics of the circuit, while multiple fault coverage may be approximated by considering the limitations on the single fault test patterns to detect multiple faults as well. Obviously, the above parameters are closely related, that is, the ability of a single fault test set to cover multiple faults decreases drastically as the number of redundant faults increases.

In the following section, a simple method for testing irredundant two level AND_OR networks is presented.

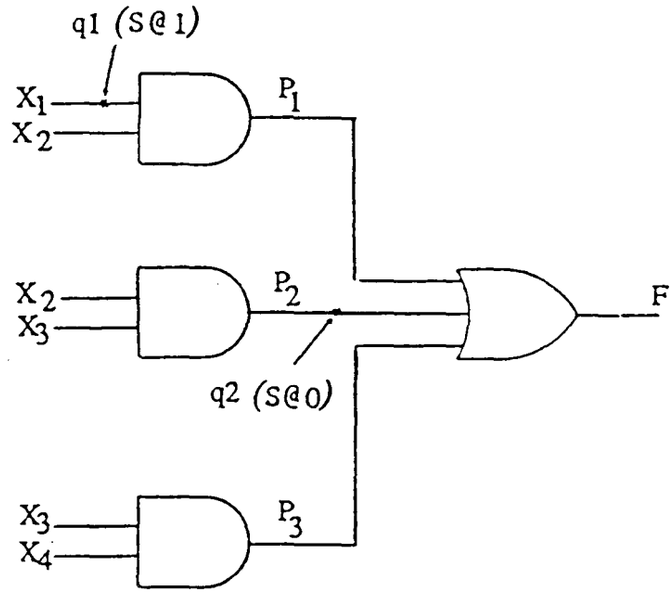
The method is shown to be a good vehicle for analyzing and predicting the effects of undetectable faults in PLAs.

3.1 PRIME IMPLICANT METHOD FOR TEST GENERATION

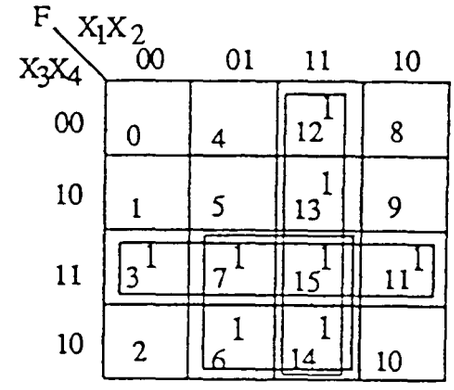
KOHAVI [13] has shown how a minimal set of tests for an irredundant 2_level AND_OR network may be derived from the set of prime implicants of the function under consideration. Based on the classical stuck at fault model, a complete test set can be generated without analyzing the topology of the circuit. Sketch (a) of Figure 3.1 shows an example of an irredundant 2_level AND_OR circuit. Karnaugh Map representation is shown in sketch (b). It is well known that for a fanout free combinational circuit, any set of tests which detects all stuck faults on primary inputs will detect all stuck faults in the rest of the circuit [1]. Thus, only stuck at ZERO (S@0) and stuck at ONE (S@1) faults on input leads of the AND gates need to be considered.

It is obvious that the AND gates in Figure 3.1(a) have one to one correspondence with the prime implicants. Thus, a stuck at fault in an AND gate will reveal some functional change in the corresponding prime implicant. This change manifests itself as a <u>growth</u> or <u>disappearance</u> of the relevant prime implicant.

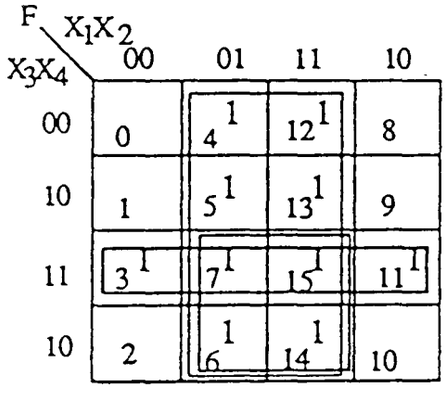
Any input to an AND gate s@1 causes a "growth" in the corresponding prime implicant. Figure 3.1(c) illustrates the effect on the prime implicant P_1 due to q_1 (X_1 S@1), i.e., P_1 (X_1X_2) grows to (X_2) causing a logical change in



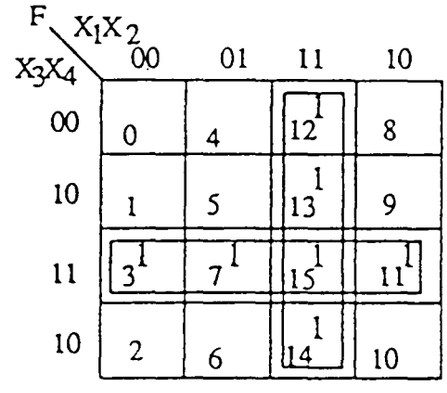
(a)



(b)



(c) Growth of P1



(d) Disappearance of P2

Fig.3.1 Prime Implicant Method for Functional Testing

the map of the output function.

Definition 3.1.1 The set of extra minterms contributed by a growth fault is called the growth term.

Obviously, there are n growth terms (or faults) associated with the prime implicant which is corresponding to an n input AND gate.

Definition 3.1.2 Any minterm that is covered by a growth term and does not belong to the function under consideration is called a free minterm.

Since the prime implicant grows to contain a growth term, then any free minterm that is covered by the growth term will detect this particular growth fault. It can be seen from Figure 3.1(c) that any minterm that belongs to the set { 4,5 } is qualified as a test for q_1. Hence, a possible minimal growth test set for the circuit example given in Figure 3.1(a) could be { 5,10 }.

On the other hand, if a s@0 fault occurs on an AND gate output, then it affects the behavior of the network as if the corresponding prime implicant was deleted from the map of the output function. This effect is shown in Figure 3.1(d) where the fault q_2 (s@0) causes the prime implicant P_2 to vanish.

Definition 3.3.3 For a given output function, any minterm that is covered by a prime implicant is said to be unique if it is not covered by any other prime implicant of the function under consideration. Otherwise, it is said to be bound.

Clearly, every unique minterm belonging to some implicant is qualified as a test for the existence of the relevant implicant. Thus, choosing a unique minterm 'arbitrary' from each prime implicant in the circuit example of Figure 3.1 yields the complete disappearance test set { 12,3,6 }.

The minimal test set to detect all single stuck at faults for the above circuit is the union of the growth and disappearance test sets; that is { 3,5,6,10,12 }.

Obviously, the above testing method is not applicable to a general PLA structure since it does not account for all possible contact faults. For instance, shrinkage and appearance faults are not covered by such method. Nevertheless, this method may be expanded, using decimal codes for the minterms, to identify redundant faults and evaluate their masking effects on normally detected faults in PLAs. Therefore, an analytic program can be designed to evaluate a difficulty measure for testing a PLA without analyzing the topology of the array structure. In this Chapter, the theoretical concept for such program, referred to as FACTPLA : Functional Analysis and the Complexity of Testing PLAs, will be established. The described approach is shown to be technology invariant and applicable to the folded versions of a PLA. The mathematical notion of the set theory is used to describe the formal aspects of the program, and some of the operations on sets that are used in this Chapter are given in Appendix A.

3.3 FUNCTIONAL LEVEL CHARACTERIZATION

In some cases, it may be possible to make use of the functional characteristics of a general digital circuit in order to explore some of its structural properties. For instance, the regularity of a PLA structure may introduce some useful properties at a higher level. The similarity between PLAs and the familiar sum of products expressions is used for this purpose. For a general sum of products expression, the cubical notation is used to represent the possible binary codes, or n_tuples, of each product term in a general multidimensional space (see Appendix A). Karnaugh Map can be considered as an attempt to project this multidimensional space onto a 2_dimensional space. It is obvious that the actual dimensionality is determined by n; the number of input variables.

In the sequel, the notion of sets is used such that all the sets are assumed to be 'finite', i.e, having only a finite number of elements. The number of elements in a finite set A is called the 'cardinality' of A and is denoted by $|A|$. Furthermore, it is very convenient to assume that all the sets are subsets of a fixed universal set (denoted by U). In the context of this thesis, however, the elements of U are the decimal codes of all the minterms found in the multidimensional space determined by n. In other words

$$U = \{ x : 0 \leqslant x \leqslant 2^n - 1 \}$$

Also, the definitions of the set 'union', 'intersection', and 'proper subset' operations (denoted by $\bigcup$, $\bigcap$, and $\subset$

respectively) are given in Appendix A. Now, some basic definitions which are used in the rest of this thesis are presented.

Definition 3.3.1 A product term P is said to be of size R if it contains R minterms.

Clearly, the size of any product term is equal to 2^i, where $i \geq 0$.

Definition 3.3.2 Two minterms covered by a product term are said to be adjacent if they differ in only one bit, i.e., the difference between their decimal codes is 2^i, where $i \geq 0$.

Definition 3.3.3 Let A and B be two sets of the same cardinality such that

$A = \{ x : x \geq 0 \}$, $B = \{ y : y \geq 0 \}$.

If, for every i, element x_i in A is adjacent to element y_i in B, then set A is said to be adjacent to set B or vice versa. On the other hand, if A and B do not have the same cardinality then a 'set adjacency' operation, denoted by (ADJ), may by defined as follows :

$A\ (ADJ)\ B = \{ x_i : |x_i - y_j| = 2^k$, where y_j : the 'first' element in B adjacent to $x_i \}$, and

$B\ (ADJ)\ A = \{ y_i : |y_i - x_j| = 2^k$, where x_j : the 'first' element in A adjacent to $y_i \}$, for $k \geq 0$.

Obviously, the set {A (ADJ) B} is not necessary equal to the set {B (ADJ) A}.

Example. Let A = {0,1,3,5,6,7,8} and B = {8,10,13,21,31}.

Then A (ADJ) B = {0,5,8 }, and

B (ADJ) A = { 8,10,13,21 }

Generally, any given minterm (m) is adjacent to a set of minterms (S). This set may be generated using the following expression [24] :

$$S = m + [2^{(i-1)}][(-1)^{m/2^{(i-1)}}] \quad \text{--------} \quad (1)$$

for i=1,2,..n where

n : the number of the input variables which define the multidimensional space containing m, and

$m/2^{(i-1)}$ is defined as an integer divide.

Recall the possible modifications that are apt to occur for a product term due to some physical failure, the following property hold.

property 3.3.1 For any product term P of size R, there are

(i) $n - Log_2 R$ possible Growth faults (= Vanish faults)

(ii) $2Log_2 R$ possible Shrinkage faults.

where n is the number of input variables.

proof

(i) A product term P of size R may grow to contain a growth term g_t of size R also. Each minterm in g_t is adjacent only to one minterm in the original product term P. Now, if R=1 (canonical term), then the number of possible growth terms (faults) in P = the number of all possible adjacent minterms = n. Thus, for a given value of n, if R increases, the number of the growth terms (or faults) decreases. This relationship can be described by the following Table

i	$R = 2^i$	n: 1	2	3	4	5	. . .	
0	1	1	2	3	4	5	. . .	
1	2	-	1	2	3	4	. . .	
2	4	-	-	1	2	3	. . .	
3	8	-	-	-	1	2	. . .	
4	16	-	-	-	-	1	. . .	<-- possible growth

Since each entry in the above Table represents a number of growth faults, then it is obvious that

the number of growth faults = n - i -------- (2)

Now, $R = 2^i$, then

$Log_2 R = i$ -------- (3)

from (2) and (3) we have

the number of growth faults = $n - Log_2 R$.

(ii) In this case, the number of shrinkage faults in any product term does not depend on the dimensionality of the space defined by n. A product term P of size R may shrink to half of its original size. The shrunk term (the rest of the minterms in P) constitutes an implicant of size R/2. Hence, the total number of possible shrunk terms (or shrinkage faults) in P can be arranged in the following Table :

i	$R=2^i$	possible implicants of size R/2
1	2	2
2	4	4
3	8	6
4	16	8

From this Table, it is easy to realize that the total number of possible shrinkage faults in P = 2 * i = $2Log_2 R$.

It should be noticed that for i=0 the product term contains only one minterm, i.e., P contains the maximum number of literals. This means that all input variables are personalized in this product term, and this case is excluded from the definition of shrinkage faults.

The above property is general to a PLA structure specified as a set of n_tuple cubes. If, for a given product term cube, the number of the X_component is k, then the size R (as defined above) of this cube is 2^k (hence, $k = \text{Log}_2 R$). By definition, shrinkage faults occur due to the incorrect connections of the unpersonalized (X_component) input variables (complement or uncomplemented) to some product line.

Thus, all possible shrinkage in the product cube =
2 * the number of X_components = $2 * \text{Log}_2 R$

Obviously, the number of all possible growth faults will be equal to the number of the non X_components. Thus, if n represents the number of all input variables to the PLA, then

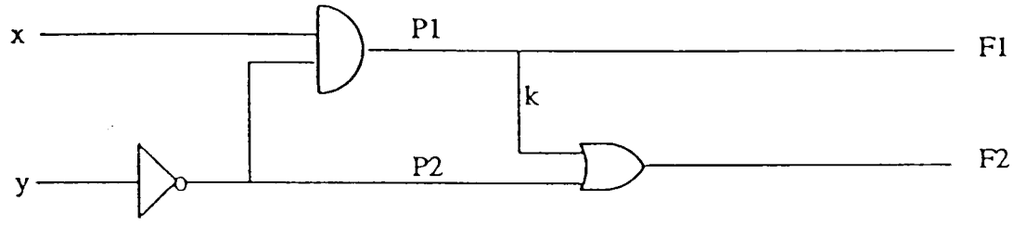
all possible growth in the product cube = $n - k$
$= n - \text{Log}_2 R$

Example. let P_i be the cube (1XX10X0X), where n = 8, K = 4. Therefore, $R = 2^k = 16$ and
possible growth faults in $P_i = n - k = 4$,
possible shrinkage faults in $P_i = 2\text{Log}_2 R = 8$.

3.4 REDUNDANT FAULTS IDENTIFICATION IN PLAs

Generally, the adoption of the crosspoint fault model in PLAs presents a clear distinction between redundant faults at the functional level. For instance, the presence of undetectable stuck_at faults in a general digital circuit can be associated with redundancy. This is shown clearly for the circuit given in sketch (a) of Figure 3.2. The stuck_at 0 fault on line k is undetectable and, if the function is implemented with a PLA, the disappearance contact fault at the junction between column f_2 and P_1 is equally undetectable (sketches b and c). Now, if the line k is removed, the resulting circuit is irredundant. However, removing the connection (device) between f_2 and P_1 results in one undetectable appearance fault. Thus, the circuit is 'logically' irredundant while the PLA is contact redundant.

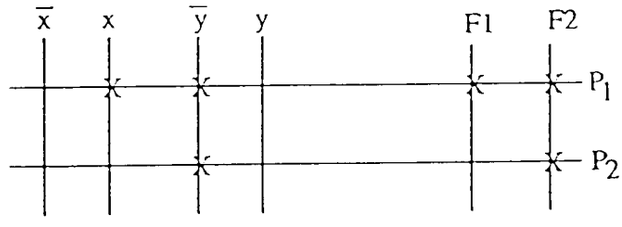
Now, the product terms of a general PLA are not restricted to be prime implicants. However, we do assume that all of the product terms are <u>essential</u> with respect to some output function. Thus, deleting any product term from the sum of products expression is guaranteed to cause a logical change in the map of the output function(s). If the PLA does not have any redundant product terms, then all vanish faults would be detectable because, if a product term vanishes, then at least one output function would be affected, provided the product term is not redundant. Redundant product terms will be assumed to have been removed from the array. Similarly, in a multiple



(a)

	x	y	F1	F2
P1	1	0	1	1
P2	-	0	0	1

(b)



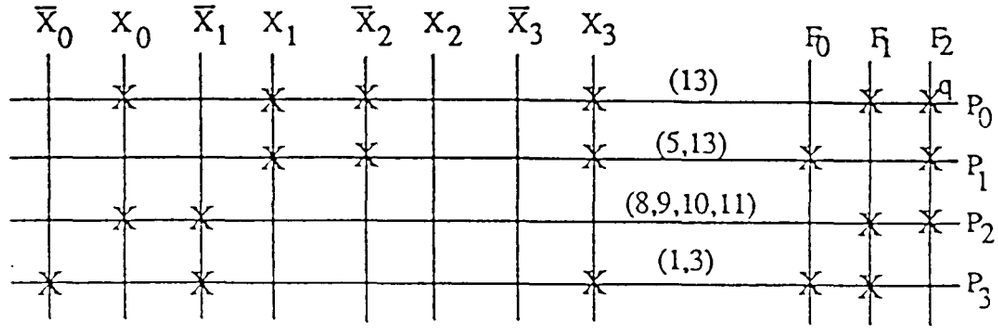
(c)

Fig. 3.2 Effects of a Redundant Function

output PLA any contact in the OR plane causing an undetectable disappearance fault can be removed without changing the function of the array. For instance, the PLA structure shown in Figure 3.3 has an obvious redundant contact at the junction between the product line P_0 and the output column f_2. It can easily be realized that this particular contact causes P_0 to be redundant with respect to f_2. Hence, the function realized by this PLA will remain the same with or without this contact.

The removal of such contacts can easily be undertaken by inspecting the maps of the output functions individually. Accordingly, each fault belonging to the union set of all vanish and disappearance faults is guaranteed to be detected by 'any' complete single contact fault test set. In this context, a complete single fault test set is assumed to contain a test pattern for every detectable single fault in the circuit under consideration. Therefore, only redundant growth, shrinkage, and appearance faults need to be considered.

In the following subsections, the basic theoretical concept for identifying redundant faults in PLAs is presented. The method is based on manipulating the decimal representation of the product terms. Two parameters associated with every product term in the PLA are suggested. The adjacent Table and the partitions of a product term are defined to be the vehicle for the analysis presented in the rest of this thesis. These parameters represent a new view for the use of the product



F_0

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

F_1

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

F_2

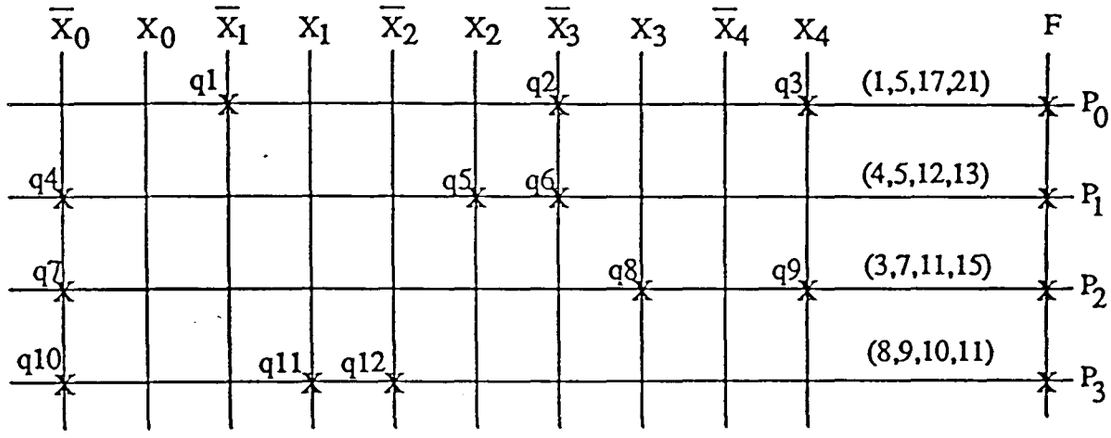
0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

Fig. 3.3 Redundant Contact (q) in the OR Plane

term fault model presented in the previous Chapter. In the sequel, the 'bound', 'unique', and 'free' minterms will be used as they defined in section 3.2.

3.4.1 Redundant Growth Faults

Based on property 3.3.1(i) of the previous section, it is possible to establish a numerical Table for any product term P such that, every column in the table represents a possible "growth fault". Obviously, every row of the Table represents the set of all minterms adjacent to a particular minterm in P. This Table will be called the "Adjacent Table" for the product term P. Figure 3.4 shows a simple PLA and the relevant Tables. In this Figure, column q1 in the table of P_0 represents the growth term of P_0 due to the missing contact fault q1, column q2 represents the growth term of P_0 due to the missing contact fault q2, and so on. Two different types of tagging are used to distinguish between the entries (minterms) of the adjacent Table. The 'circled' minterms in the Table are those belonging to the output function under consideration, while the minterms tagged with an 'asterisk' (on the left side of the Table) represent bounded minterms with respect to some output function. It should be noticed that for different output functions, the structure (in terms of the tagging) of the adjacent Tables will be different. The notion of the adjacent Table may be derived formally using property 3.3.1 with the aid of the following definition.



P0	q3	q2	q1
1	0	③	⑨
5*	④	⑦	⑬
17	16	19	25
21	20	23	29

P1	q6	q5	q4
4	6	0	20
5*	⑦	①	㉑
12	14	⑧	28
13	15	⑨	29

P2	q9	q8	q7
3	2	①	19
7	6	⑤	23
11*	⑩	⑨	27
15	14	⑬	31

P3	q12	q11	q10
8	⑫	0	24
9	⑬	①	25
10	14	2	26
11*	⑮	③	27

Fig. 3.4 Adjacent Tables for a Simple PLA

Definition 3.4.1 If A and B are sets, the complement of B in A , written as A - B, is defined by

$$A - B = \{ x : x \in A , x \notin B \}.$$

Now, if P is a product term of size R, then a set $(A)_p$ may be defined as follows

$(A)_p = \{ x_i : x_i$ is a decimal code for some minterm in P $\}$ such that $x_i > x_j$ for $i>j$ and $1 \leqslant i,j \leqslant R$.

Also, let $(S)_p$ be a set whose elements are themselves sets written as

$$(S)_p = (\bigsqcup_{i=1}^{n} s_i)_p = s_1 \bigcup s_2 \cdots \bigcup s_n$$

where s_i : a possible set that is adjacent to $(A)_p$ (see definition 3.3.3),

n : number of input variables.

The set s_i is generated using the following expression :

$$s_i = x_j + [2^{(i-1)}][(-1)^{x_j/2^{(i-1)}}] \quad \text{--------} \quad (4)$$

for $1 \leqslant j \leqslant R$, $x_j \in (A)_p$, and $x_j/2^{(i-1)}$ is defined as an integer divide.

From property 3.3.1, if R>1 then some of the s_i's sets must be 'equal' to $(A)_p$. Accordingly, the Adjacent table of P is the set $(AT)_p$ where

$$(AT)_p = (S)_p - (A)_p$$

$$= (\bigsqcup_{i=1}^{n} s_i) - (A)_p \quad \text{--------} \quad (5)$$

Obviously, $(A)_p - (A)_p = \Phi$ (the empty set). Thus, the actual number of the sets that are generated by expression (5) is limited by n and it is equal to $(n - \log_2 R)$, as it has been proved in property 3.3.1. Hence, expression (5) becomes

$$(AT)_p = gf_1 \bigcup gf_2 \cdots \bigcup gf_{n-\log_2 R} = \bigcup_{j=1}^{n-\log_2 R} gf_j \quad \text{--------} \quad (6)$$

where

$gf_i = s_i - (A)_p \neq \Phi$

= the fault set for some growth fault in P.

Now, since the columns of an adjacent Table represent growth fault sets, then any <u>free</u> (uncircled) minterm in each column is qualified to be a test vector for the relevant growth fault. In other words, if F represents the set of all product terms belonging to the function under consideration, i.e.,

$$F = \sum_{i=1}^{m} (A)_{pi} \subset U \quad (U : \text{the universal set}) \quad \text{--------} \quad (7)$$

then any fault set satisfying the condition

$$(gf)_{Pi} - (F - (A)_{Pi} = \Phi \quad \text{--------} \quad (8)$$

represents a redundant growth fault in P_i. For instance, if the contact at q8 on P_2 of Figure 3.4 is missing incorrectly, then $(A)_{P2}$ grows to contain the fault set (1,5,9,13). Column q8 of the Table of P_2 shows that no minterm is free and, hence, the fault is undetectable.

It is worth noting that the order of the columns in a given Table has some similarity with the physical locations of the contact faults on the corresponding product line. It is easy to realize that the first column represents the last missing contact defect in the product line, the second column represents the defect next to the last one, and so on. Fortunately, such arrangement, which is very attractive for locating the redundant growth faults in a PLA, represents the exact way with which expression (1) of section 3.3 works.

The concept of the growth Table may also be used to generate the complete set of single growth fault test patterns in PLAs. Later on in the Chapter, the policy of chosing more realistic test patterns for multiple fault coverage in PLAs is described.

3.4.2. Redundant Shrinkage Faults

An extra contact fault at some unpersonalized input variable in a product line causes the corresponding product term to shrink to half of its original size. A simple heuristic will be used to represent all possible shrunk terms (the rest of minterms left due to a shrinkage fault). This will be done simply by ordering the decimal codes of the minterms constituting a product term in an ascending manner. Recall the adjacent relationships in the n_dimensional space, it is easy to realize that for a product term P of size R, all possible shrunk terms may be obtained by a simple partitioning process. The number of the partitions in P is determined by its size R.

If $R = 2^i$, $i = 1,2,3...$, then the number of the partitions in P is equal to i. Indeed, each partition contains a pair of blocks; one represents the shrunk term and the other represents the fault responsible for it. In other words, each block is equivalent to an extra contact fault at one of the two columns of an input variable X_i, where X_i is unpersonalized in P. Figure 3.5 illustrates the partitioning process and the resultant blocks for different product terms. Now, if $(A)_p$ is defined in manner given in the previous subsection, then the set of partitions (PT) in the term P may be described below :

$$(PT)_p = \bigsqcup_{i=0}^{\text{Log}_2 R - 1} I_i \qquad \text{--------- (9)}$$

where R : the size of P, and

$I_i = B_{i1} \bigcup B_{i2}$ the two blocks (or fault sets) of partition i.

In Mathematical terms, each partition 'decomposes' $(A)_p$ into two non_empty disjoint subsets (or blocks) of the same 'cardinality' (number of elements). The derivation of the first block in partition i from $(A)_p$ is given below (where i = 0 being the first partition) :

$$(B)_i = \bigsqcup_{k=1}^{2^i} b_{ik} \quad \text{where } b_{ik} = [x_{start \longrightarrow limit} : x \in (A)_p]$$

such that

$$start = j(R/2^i)$$

$$limit = (1/2 + j)(R/2^i) - 1$$

for $0 \leqslant j \leqslant 2^i - 1$

(a) $(A)_p = \{ 0,1,2,3 \} \longrightarrow R = 4 = 2^2$

then P is partitioned as follows

$PT_0 = (0,1)\ (2,3)$

$PT_1 = (0,2)\ (1,3)$

(b) $(A)_p = \{ 0,1,2,3,16,17,18,19 \} \longrightarrow R = 8 = 2^3$

then P is partitioned as follows

$PT_0 = (0,1,2,3)\ (16,17,18,19)$

$PT_1 = (0,1,16,17)\ (2,3,18,19)$

$PT_2 = (0,2,16,18)\ (1,3,17,19)$

(c) $(A)_p = \{ 4,5,6,7,12,13,14,15,20,21,22,23,28,29,30,31 \} \longrightarrow R = 16 = 2^4$

then P is partitioned as follows

$PT_0 = (4,5,6,7,12,13,14,15)\ (20,21,22,23,28,29,30,31)$

$PT_1 = (4,5,6,7,20,21,22,23)\ (12,13,14,15,28,29,30,31)$

$PT_2 = (4,5,12,13,20,21,28,29)\ (6,7,14,15,22,23,30,31)$

$PT_3 = (4,6,12,14,20,22,28,30)\ (5,7,13,15,21,23,29,31)$

Fig. 3.5 Examples of the Product Term Partitions

The above partitions will be used to diagnose all redundant shrinkage faults in a PLA. But first, two types of product terms must be defined :

definition 3.4.1. A product term P_i is said to be non-isolated (with respect to a given output function) if it contains at least one minterm which is covered by some other product term P_j $(i \neq j)$ of the function under consideration. Otherwise, P_i is said to be an isolated product term.

Again, if R represents the size of the term P, then it is possible to obtain the set of shrinkage tests for P according to one of the following cases :

case(i) R = 1 : in this case, $(A)_p$ contains only one element that is responsible to detect any extra contact fault on the product line P.

case(ii) R = 2 : if P is isolated, then both elements of $(A)_p$ will constitute a complete shrinkage test for P. However, if P is non_isolated, then at least one of the elements in $(A)_p$ must be a 'unique' minterm and, hence, it will detect only one of the possible two shrinkage faults in P.

case(iii) R > 2 : in this case, the set of shrinkage tests for P may be derived from one of the following pairs :

$$(x_{1+j}, x_{R-j}) \qquad \text{for } 0 \leqslant j \leqslant R/2 - 1$$

where $x_i \in (A)_p$ and $1 \leqslant i \leqslant R$

Figure 3.6 illustrates the concept of these pairs for a general product term, while the rational justification of

Let $(A)_p$= { 0,1,2,3,16,17,18,19 }

(i) P is isolated

then possible shrinkage tests for P

(0,19) or (1,18) or (2,17) or (3,16)

(ii) P is non_isolated

then possible pairs are

(0,19) ,(1,18) , (2,17) , and (3,16)

(a) assume that 1, 3, 16, and 17 are bounded minterms then

a 'complete' shrinkage test for P could be (0,19)

(b) assume that 0, 1, 3, and 17 are bounded minterms then

a 'complete' shrinkage test for P coul be (19,2,16)

(c) assume that 0, 1, 2, and 3 are bounded minterms then

a possible shrinkage test for P could be (19,16) or (18,17)

(not complete)

(d) assume that 1, 3,17 , and 19 are bounded minterms then

a possible shrinkage test for P could be (0,18) or (2,16)

(not complete)

Fig. 3.6 Shrinkage Test existence for a Product Term

using them is given below :

1. P is isolated : then all the minterms belonging to P are unique. Thus, any minterm in P is qualified as a test for some shrinkage faults. In fact, every unique minterm detects half of all possible shrinkage faults in P. This is easily verified from the partitions of P. In Figure 3.5, every minterm is belonging to only one block (or shrinkage fault set) of each partition and, hence, is qualified to detect half of the total shrinkage faults. Furthermore, the partitions show that any two minterms chosen to detect all possible shrinkage faults in P must constitute one of the pairs given above. Accordingly, for an isolated product term, 'any' pair from the set (x_{l+j}, x_{R-j}), $0 \leqslant j \leqslant R/2 - 1$, is qualified as a complete shrinkage test for that term (see Figure 3.6(i)).

   Obviously, in a PLA, no redundant shrinkage faults exist on any row carrying an isolated product term.

2. P is non isolated : in this case, at least one of the minterms of P must be bound. Generally, it may not be possible to find a pair of shrinkage tests in the manner given for the isolated product term. However, one of the minterms of P should be unique and hence belongs to one of the above pairs. Therefore, any given pair (x_a, x_b) such that one of the pair's component, say x_a, is unique, then if the other component x_b is also unique, P will have a complete

shrinkage test and no redundant shrinkage faults (case (a) in Figure 3.6). However, if x_b is bounded for all possible pairs, then any two unique minterms, x_{b1} and x_{b2}, of P which are adjacent to x_b will constitute a complete test set for the other half of shrinkage faults in P (case (b) in Figure 3.6). The validity of this point can be easily verified by considering the resultant partitions brought out by the shrinkage faults in terms of K_map representation.

If the above condition does not hold for any pair, then no complete shrinkage test exists for P and it must contain some redundant shrinkage faults. These faults are redundant with respect to any test set.

It follows immediately that all redundant shrinkage faults can be identified by a simple inspection of the partition blocks shown in Figure 3.5. For a given partition, if all minterms belonging to one block of the partition are bounded, then the other block represents a shrunk product term caused by a 'redundant' shrinkage fault. Let F be defined in the manner given in expression (7) and V be the set of all common elements, i.e.,

$$V = (A)_{P1} \bigcap (A)_{P2} \cdots \bigcap (A)_{Pm} = \prod_{i=1}^{m} (A)_{Pi} \quad \text{---- (10)}$$

Then, any block, say the first block B_1, in partition j of the product term P_i satisfying the condition

$$(B_{j1})_{Pi} - V = \Phi \quad \text{-------- (11)}$$

represents a redundant shrinkage fault in P_i.

The locations of redundant shrinkage faults in a general product line P are determined directly from the set of partitions of P. The Partitions are arranged such that the first partition represents the first unpersonalized (X_component) input in the cube of the product term, and its first block represents an extra contact fault at the complemented input column, while its second block represents an extra contact fault at the uncomplemented input column.

3.4.3 Redundant Appearance Faults

For a multiple output PLA, appearance faults must be considered, in addition to shrinkage and growth faults. Appearance faults are caused by an unnecessary presence of extra contacts in the OR plane of the PLA. Consider a product term P_i (i^{th} row), which belongs to the function f_j (j^{th} output column) but does not belong to f_k (k^{th} output column). Now, under an extra contact fault at the junction of the i^{th} row and k^{th} column, the product term P_i appears on the map of the output function f_k.

Obviously, any minterm which is covered by P_i but not f_k qualifies as a test pattern for the appearance fault in question.

In the following section, the policy of using the adjacent Tables and partitions for evaluating the effects of masking among faults in PLAs is presented.

3.5 MASKING INFLUENCES ON MULTIPLE FAULT COVERAGE

A multiple fault is detected if at least one of its components is detected [29]. Figure 3.7 shows a bridging fault between two output columns in a nMOS PLA. Assume that the short is ORed, then it will be equivalent to the multiple contact fault $q_1 q_2 q_3$ in the OR plane of the PLA. If one of the fault component (q_1 or q_2 or q_3) is testable, then the multiple fault is testable by a test for such component.

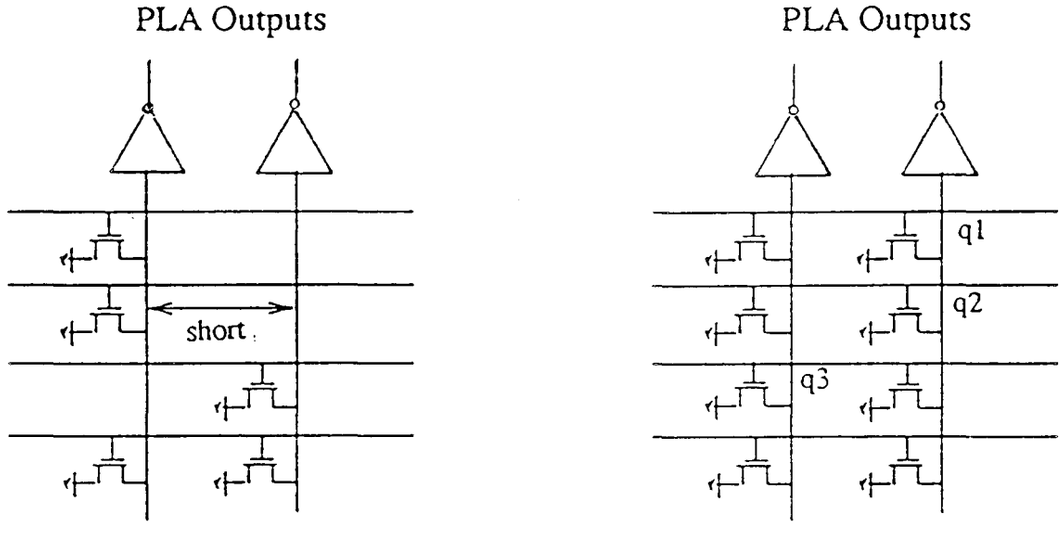
Accordingly, to guarantee the detection of a multiple fault, at least one of its fault components should not be masked by any redundant fault in the circuit. The concept of fault masking is further elaborated below.

Let T_α be the set of all possible test patterns for the single fault α in a general digital circuit (note that T_α may contain only one pattern). If α is proceeded by some "masking" fault β, then two different cases have to be considered :

(a) fault β masks fault α for <u>all</u> the patterns in T_α. This masking phenomenon, written as β_α, may be represented by the following set

$$M = \max \{ (\beta_\alpha)_{t_i} : t_i \in T_\alpha \} \quad \text{for} \quad 1 \leqslant i \leqslant |T_\alpha|$$

Then β_α is 'true' if and only if β is undetectable. Therefore, a difficulty measure (MASK) may be defined to mark the existence of every such condition. If k represents the number of the undetectable faults, L represents the number of the testable single faults in a



(a) nMOS PLA with a Short Fault

(b) Equivalent Faulty Circuit

Fig. 3.7 Effects of Bridging Faults in PLAs

given circuit, then

$$(MASK)_{\beta} = \sum_{(\beta_{\alpha_j})_{j=1}}^{L} j \qquad \text{-------- (12)}$$

$$(MASK)_{cct} = \sum_{i=1}^{K} \sum_{(\beta^i_{\alpha_j})_{j=1}}^{L} j \qquad \text{-------- (13)}$$

(b) fault β masks fault α for <u>some</u> patterns in T_α. Obviously, the simultaneous fault $\alpha\beta$ could be detected by some patterns in T_α even if β was undetectable. However, this type of masking has less restriction than the one described in case(a). hence, if D is a 'proper subset' (see Appendix A) from the set M described above, then

$$D = \{(\beta_\alpha)_{t_i} : t_i \in T_\alpha\} \subset M \quad \text{for} \quad 1 \leqslant i \leqslant |D|$$

Therefore, β_α is 'true' if and only if β is undetectable fault. In general, a difficulty measure (RISK) may be defined to count the patterns in T_α under which β masks α. Again, if K represents the number of the undetectable faults, L represents the munber of the detectable faults in the circuit, then

$$(RISK)_{\beta} = \sum_{i=1}^{L} \sum_{(\beta_{\alpha_i})_{j=1}}^{|D|} j \qquad \text{-------- (14)}$$

$$(RISK)_{cct} = \sum_{i=1}^{k} \sum_{e=1}^{L} \sum_{(\beta^i_{\alpha_e})_{j=1}}^{|D|} j \qquad \text{-------- (15)}$$

At the functional level, the key concept in evaluating the above measures is to consider first the effect of the

masking fault. This strategy is vital if the masking fault was undetectable.

3.5.1 Masking Evaluation in a Single Output PLA

Consider the simple PLA structure given in Figure 3.8. Obviously, the adjacent tables and the partitions indicate that the structure is irredundant, i.e., contains no redundant faults. Nevertheless, this structure may be used to illustrate the effect of fault masking in a general PLA.

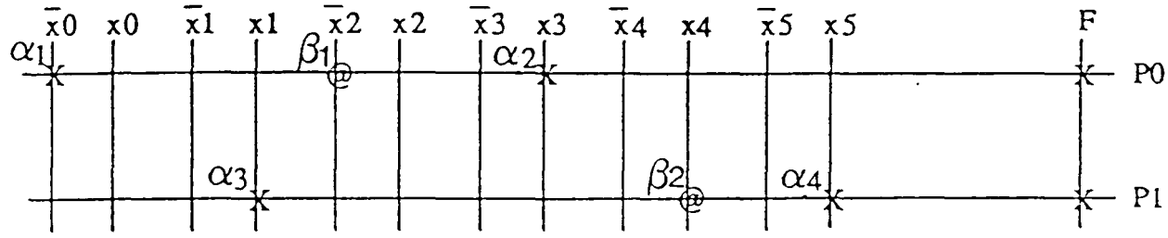
In Figure 3.8, for the missing contact fault $\alpha 1$ on P_0, any pattern in the set $T_{\alpha 1}$, where

$$T_{\alpha 1} = \{ 36,37,38,39,44,45,46,47,52,54,60,62 \} \subset gf_{\alpha 1}$$

is qualified to detect $\alpha 1$. Now, the extra contact fault $\beta 1$ on the same product line masks the detection of $\alpha 1$ for some tests in $T_{\alpha 1}$. The effect of this phenomenon is determined from the masking fault ($\beta 1$ in this case) in the following manner. Due to $\beta 1$, the product term P0 shrinks to

$$[(A)_{P0}]_{\beta 1} = \{ 4,5,6,7,20,21,22,23 \}$$

Then the masking should be represented by the combined effect of both faults such that the effect of $\beta 1$ dominates the effect of $\alpha 1$. In other words, $T_{\alpha 1}$ should be reduced according to the reduction in the adjacent Table of P_0 caused by the fault $\beta 1$. In this context, the 'set adjacency' operation, defined in section 3.3, may well be used to express this masking dominancy. Accordingly, only the patterns in the set



P0	$gf_{\alpha 2}$	$gf_{\alpha 1}$
4	0	36
5	1	37
6	2	38
7	3	39
12	8	44
13	9	45
14	10	46
15	11	47
20	16	52
21*	(17)	(53)
22	18	54
23*	(19)	(55)
28	24	60
29*	(25)	(61)
30	26	62
31*	(27)	(63)

P1	$gf_{\alpha 4}$	$gf_{\alpha 3}$
17	16	1
19	18	3
21*	(20)	(5)
23*	(22)	(7)
25	24	9
27	26	11
29*	(28)	(13)
31*	(30)	(15)
49	48	33
51	50	35
53	52	37
55	54	39
57	56	41
59	58	43
61	60	45
63	62	47

Adjacent Tables

partitions of P0

[(4,5,6,7,12,13,14,15) (20,21*,22,23*,28,29*,30,31*)]

[(4,5,6,7,20,21*,22,23*) (12,13,14,15,28,29*,30,31*)] ← β_1

[(4,5,12,13,20,21*,28,29*) (6,7,14,15,22,23*,30,31*)]

[(4,6,12,14,20,22,28,30) (5,7,13,15,21*,23*,29*,31*)]

partitions of P1

[(17,19,21*,23*,25,27,29*,31*) (49,51,53,55,57,59,61,63)]

[(17,19,21*,23*,49,51,53,55) (25,27,29*,31*,57,59,61,63)]

[(17,19,25,27,49,51,57,59) (21*,23*,29*,31*,53,55,61,63)]

[(17,21*,25,29*,49,53,57,61) (19,23*,27,31*,51,55,59,63)] ← β_2

Fig. 3.8 Effects of Fault Masking in PLAs

$$[T_{\alpha 1}]_{\beta 1} = T_{\alpha 1} \text{ (ADJ) } [(A)_{P0}]_{\beta 1}$$
$$= \{ 36,37,38,39,52,54 \}$$

can detect the simultaneous fault $\alpha 1 \beta 1$.

On the other hand, the product term P1 shrinks to

$$[(A)_{P1}]_{\beta 2} = \{ 19,23,27,31,51,55,59,63 \}$$

due to the extra contact fault $\beta 2$. From the partitions of P1, any pattern in the set

$$T_{\beta 2} = \{ 17,25,49,53,57,61 \}$$

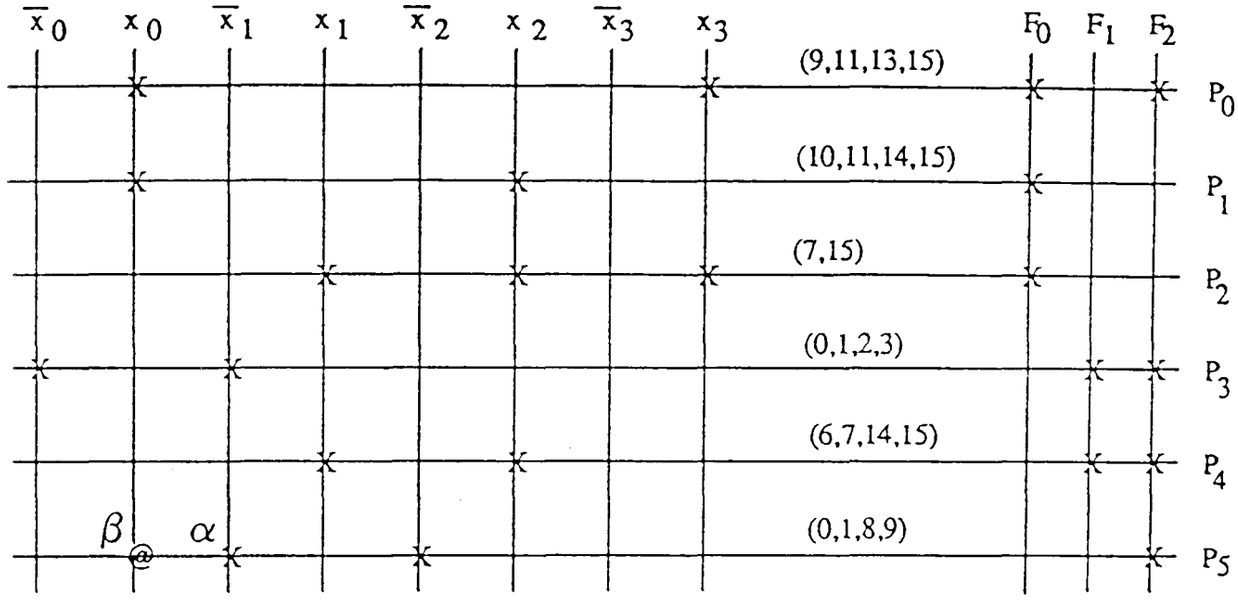
is qualified to detect $\beta 2$. However, if fault $\alpha 1$ exists, then it will mask $\beta 2$ for some tests in $T_{\beta 2}$. In this case, the combined effect of $\alpha 1$ and $\beta 2$ is determined from the growth fault set of the masking fault $\alpha 1$; $gf_{\alpha 1}$. Any element in $T_{\beta 2}$ which also belongs to $gf_{\alpha 1}$ is disqualified to detect the simultaneous fault $\alpha 1 \beta 2$. It is easy to realize that the patterns in the set

$$[T_{\beta 2}]_{\alpha 1} = T_{\beta 2} \cap gf_{\alpha 1}$$
$$= \{ 53,61 \}$$

fail to detect $\alpha 1 \beta 2$.

The following examples illustrate the influences of fault masking due to 'redundant' faults in a general PLA structure.

Example 1. For the PLA structure shown in Figure 3.9(a), the extra contact fault at the junction between X_0 and P_5 causes an undetectable shrinkage fault β in P_5. The adjacent Table of P_5 shows that for the growth fault α,



F0

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

F_1

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

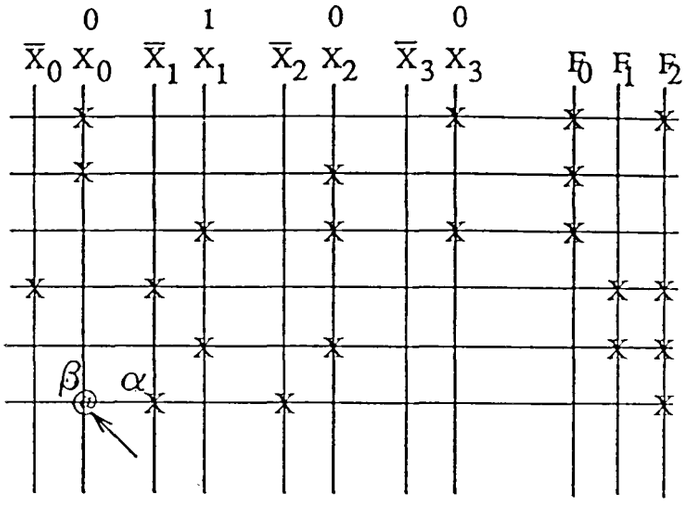
F2

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

Adjacent Table of P5 with respect to F2

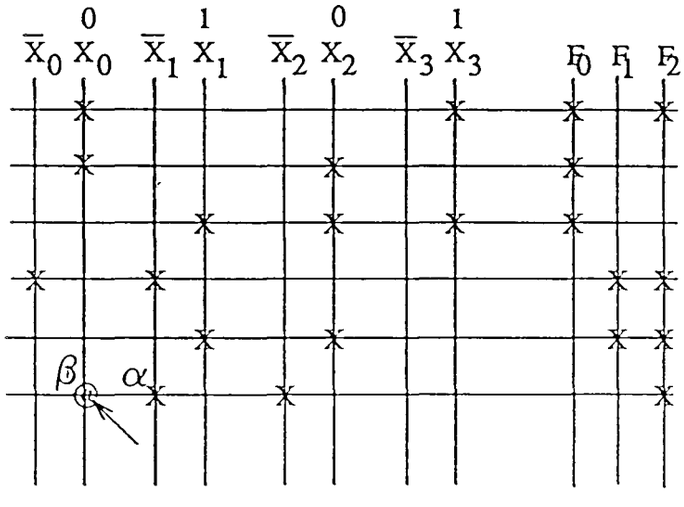
P5		α
0*	②	4
1*	③	5
8	10	12
9*	⑪	⑬

Fig. 3.9a Shrinkage Fault (β) Masks Growth Fault (α)



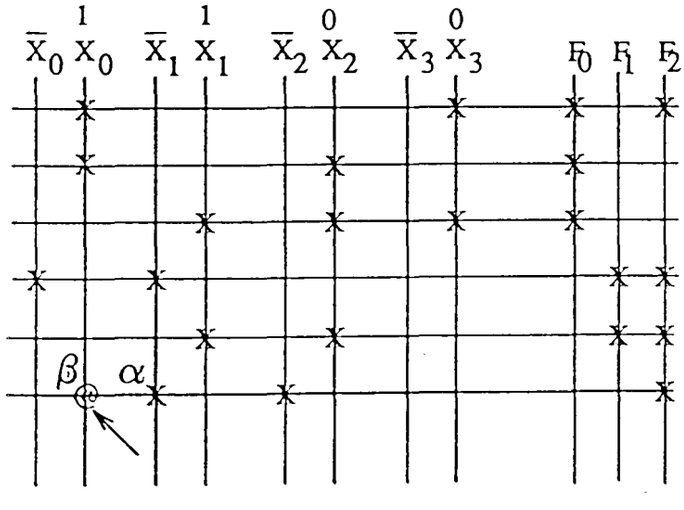
(F2)	(F2)$\alpha\beta$
0.0	0.0
+	+
-	-
+	+
-	-
+	+
1.0	1.0
+	+
1.0	1.0
+	+
0.1	0.1
0	0

no change



(F2)	(F2)$\alpha\beta$
0	0
+	+
-	-
+	+
-	-
+	+
1.0	1.0
+	+
1.0	1.0
+	+
0.1	0.1
0	0

no change



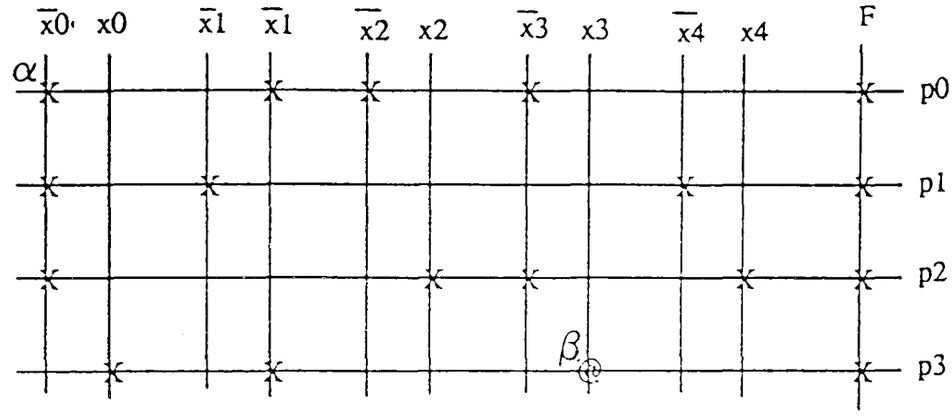
(F2)	(F2)$\alpha\beta$
0	0
+	+
-	-
+	+
-	-
+	+
0.0	0.0
+	+
1.0	1.0
+	+
0.1	1.1
0	1

change

Fig. 3.9b Effects of Fault Masking for Different Tests

the available tests are the minterms 4 (0100), 5 (0101), and 12 (1100). Assume that the single contact fault test set T_s includes the patterns (0100) and/or (0101) but not (1100), then α will not be detected by the set T_s, if the redundant fault β is also present. It is easy to realize that under the test (0100) or (0101), the value on the output F_2 will not be affected by the multiple fault $\alpha\beta$. For both tests, F_2 will have the same logic value with or without the fault $\alpha\beta$. This is illustrated in Figure 3.9(b), where the logical change in the signal values has been traced for all available tests for α. However, during the normal operation, for the input vector (1100), F_2 will have logic ZERO without the fault $\alpha\beta$ and logic ONE with the fault. Then output of the PLA will be incorrect for this input.

Example 2. For the PLA structure shown in Figure 3.10(a), α is a missing contact fault which causes P_0 to contain a redundant growth fault. For the extra contact fault β at the junction between X_3 and P_3, the available tests are the minterms 24 (11000), 25 (11001), 28 (11100), and 29 (11101). For both inputs (11000) and (11001), the presence of the multiple fault $\alpha\beta$ does not alter the logic value at the output of the PLA. This is shown clearly in Figure 3.10(b) where the output response is shown to be affected only by applying the tests (11100) or (11101). Thus, one of the two minterms, (28) or (29), should be included in T_s to ensure the detection of the multiple fault $\alpha\beta$.



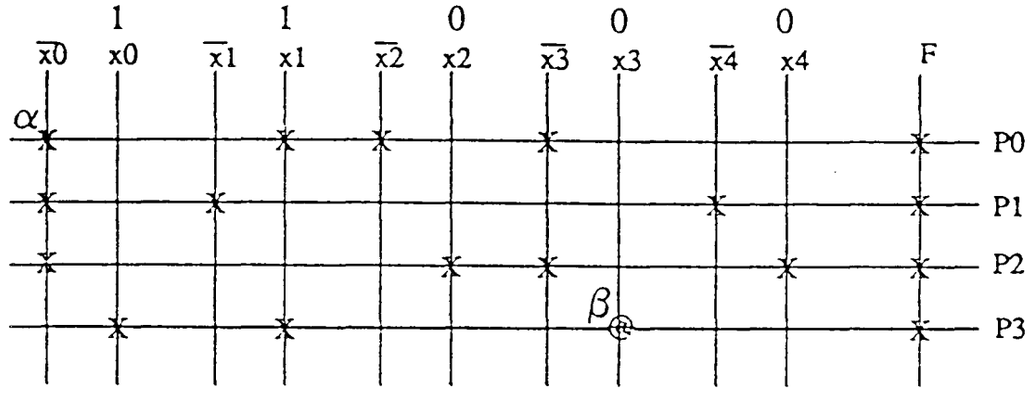
F

0	4	12	8	24	28	20	16
1	5	13	9	25	29	21	17
3	7	15	11	27	31	23	19
2	6	14	10	26	30	22	18

Adjacent Tables of P0

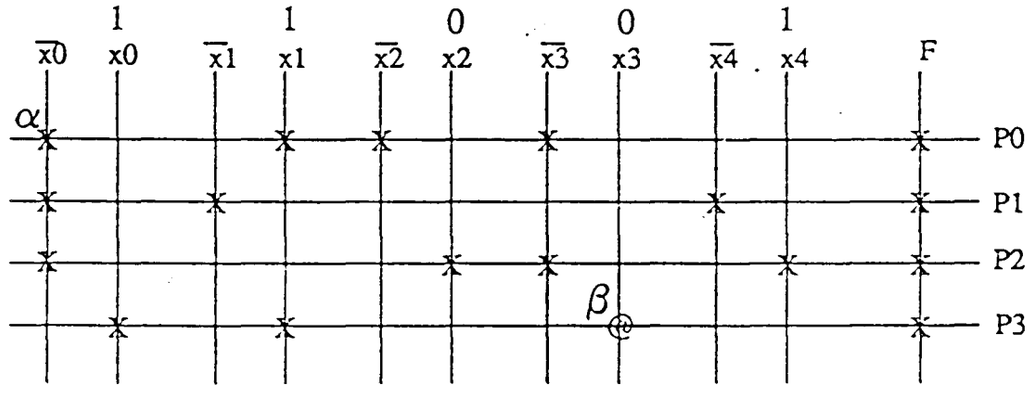
P0				α
8	10	12	(0)	(24)
9	11	(13)	1	(25)

Fig. 3.10a Growth Fault (α) Masks Shrinkage Fault (β)



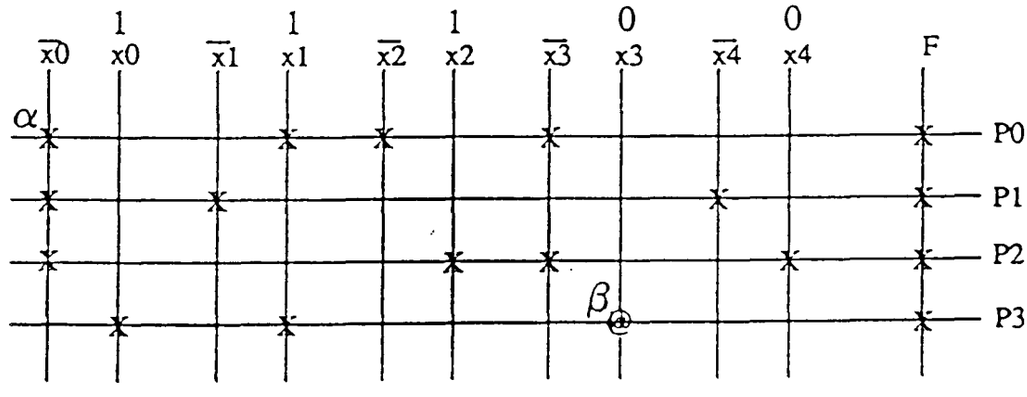
F	$F_{\alpha\beta}$
0.1.1.1	1.1.1
+	+
0.0.1	0.0.1
+	+
0.0.1.0	0.0.1.0
+	+
1.1	1.1.0
1	1

no change.



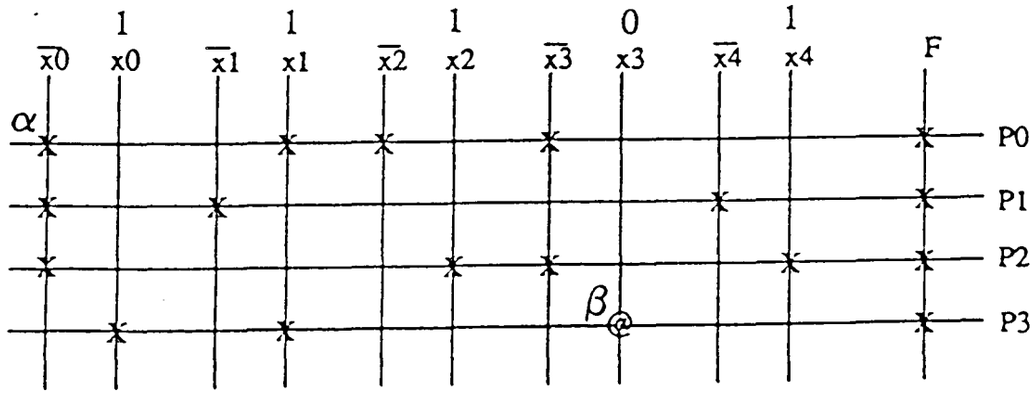
F	$F_{\alpha\beta}$
0.1.1.1	1.1.1
+	+
0.0.0	0.1.0
+	+
0.0.1.1	0.0.1.1
+	+
1.1	1.1.0
1	1

no change



F	$F_{\alpha\beta}$
0.1.0.1	1.0.1
+	+
0.0.1	0.0.1
+	+
0.1.1.0	0.1.1.0
+	+
1.1	1.1.0
1	0

change



F	$F_{\alpha\beta}$
0.1.0.1	1.0.1
+	+
0.0.0	0.0.0
+	+
0.1.1.1	0.1.1.1
+	+
1.1	1.1.0
1	0

change

Fig. 3.10b Effects of Fault Masking for Different Tests

3.5.2 Masking evaluation in a Multiple Output PLAs

The existence of appearance faults in a general PLA structure has imposed the following consequences :

(i) a detectable extra contact (shrinkage) fault in the input part of row i of a PLA may be masked (on a given test vector) by a redundant appearance fault in row j, $j \neq i$ (property 2.5.2 in Chapter 2).

(ii) a possible case of property 2.5.1 (in Chapter 2) could be interpreted as follows :
"for a given test vector, X_t, a detectable appearance fault can only be masked by a redundant shrinkage fault in the same row of the PLA".

Accordingly, the following two cases complete the evaluation of the effects of redundant faults in PLAs :

case (i) (on the same row of the PLA)

let β be a redundant shrinkage fault on row i of a (n,m,z)_PLA such that the partition of P_i due to this fault contains the two blocks B1 and B2 where

B1 = the shrunk term = $[(A)_{Pi}]\beta$, and
B2 = the fault set. (empty in this case)

For the output function f_k, $1 \leqslant k \leqslant z$, such that P_i does not belong to f_k, an extra contact fault , γ, at the junction between P_i and f_k causes the appearance of P_i on the map of the function f_k. If, due to fault β, $B1 \subset F_k$ (see expression (7) in section 3.4.3 for the definition of the set F_k), then β masks γ and the simultaneous fault $\beta\gamma$ is undetectable.

case (ii) (on different rows of the PLA)

let γ be a redundant extra contact fault at the junction between row i and the output column k of a (n,m,z)_PLA. A normally detectable shrinkage fault β in row j such that $j \neq i$ and P_j belongs to f_k only, may be masked by the fault γ. Let B_β be the fault set of β. If $B_\beta \subset (A)_{Pi}$, then the simultaneous fault $\beta\gamma$ is undetectable.

Again, it should be noticed that in the above two cases the effect of the masking fault is considered first. The combined effect of both faults is then analyzed prior to evaluate the masking effect.

In the next Chapter, the analysis of the redundant contact faults is shown to be performed by a simple inspection and manipulation of the adjacent Tables and partitions produced for each product term.

3.6 SUMMARY

A concept for evaluating the effects of redundant faults in PLAs has been presented. A new description for the product term fault model is formulated in terms of two sets associated with each term; the adjacent Table set and the set of partitions. These sets are analyzed and possible redundant faults have been shown to exist within three types of contact faults. These faults (growth, shrinkage, and appearance) can be identified and their masking influence on detectable faults may be evaluated by investigating the adjacency relationships and the properties of the output function(s). Such analysis may

be carried out before the actual derivation of any single contact fault test set (T_s) in a PLA takes place. Therefore, the complexity of testing and the ability of T_s to cover more multiple faults can be established by producing some difficulty measures for the actual fault masking in the array. In the next Chapter, the algorithmic realization of the above concepts and the estimation of the complexity of the implementation is presented.

CHAPTER FOUR

FACTPLA PROGRAM IMPLEMENTATION

4.1 INTRODUCTION

In the previous Chapter, two difficulty measures have been established and shown to have a great influence on the complexity of testing a PLA. The evaluation of these measures is embodied in a general analytic program (FACTPLA) : Functional Analysis and the Complexity of Testing PLAs. The basic program consists of two main steps : identifying the redundant contact faults and evaluating their masking effects in PLAs. For testing purposes, the distinction between two different PLA structures having the same silicon area is based on the differences between

(i) the effects of fault masking (MASK measure), and

(ii) the restrictions on single fault test patterns to cover multiple faults (RISK measure).

The above two measures have been evaluated from the parameters of adjacent Tables and partitions which are defined in Chapter 3. The measures are obtained for every redundant single contact fault. The first (MASK) measure accumulates the possible masking occurrence due to redundant single faults, while the second (RISK) measure indicates the difficulty of testing multiple faults due to the arbitrary choice of the single fault test patterns.

Indeed, a redundant fault in a general digital circuit makes testing difficult regardless that the fault is single or embodied in a more general multiple fault. Accordingly, the values of the above measures reflect the

effectiveness of any test set derived on the single fault assumption bases. In VLSI environment, predicting such effectiveness is vital due to the increasing number of multiple faults which have to be considered. In the rest of this Chapter, the algorithmic realization and application of FACTPLA program is presented, and the complexity of computation is also discussed.

4.2 FAULT DATA STRUCTURE

Obviously, the redundant contact faults in a PLA represent the framework of the analysis performed by FACTPLA program. Figure 4.1 illustrates the fault data structure constituting the bases of FACTPLA's algorithms presented in the following sections. In this Figure, a redundant contact fault on some row, say row i, of the PLA is either a shrinkage or growth fault in the AND plane, or an appearance fault in the OR plane of the array. The effects of every redundant fault are associated with the MASK and RISK measures which reflect the difficulty (of testing) imposed by the fault in question. The 'MASK_RISK' relationships depend mainly on the location of the redundant fault in the array structure.

Note that disappearance faults are excluded form the structure shown in Figure 4.1. In the previous Chapter, redundant disappearance faults are shown to have no effects on other faults in a PLA (see Figure 3.3). The removal of the contacts (devices) causing such faults would not change the functionality of the array.

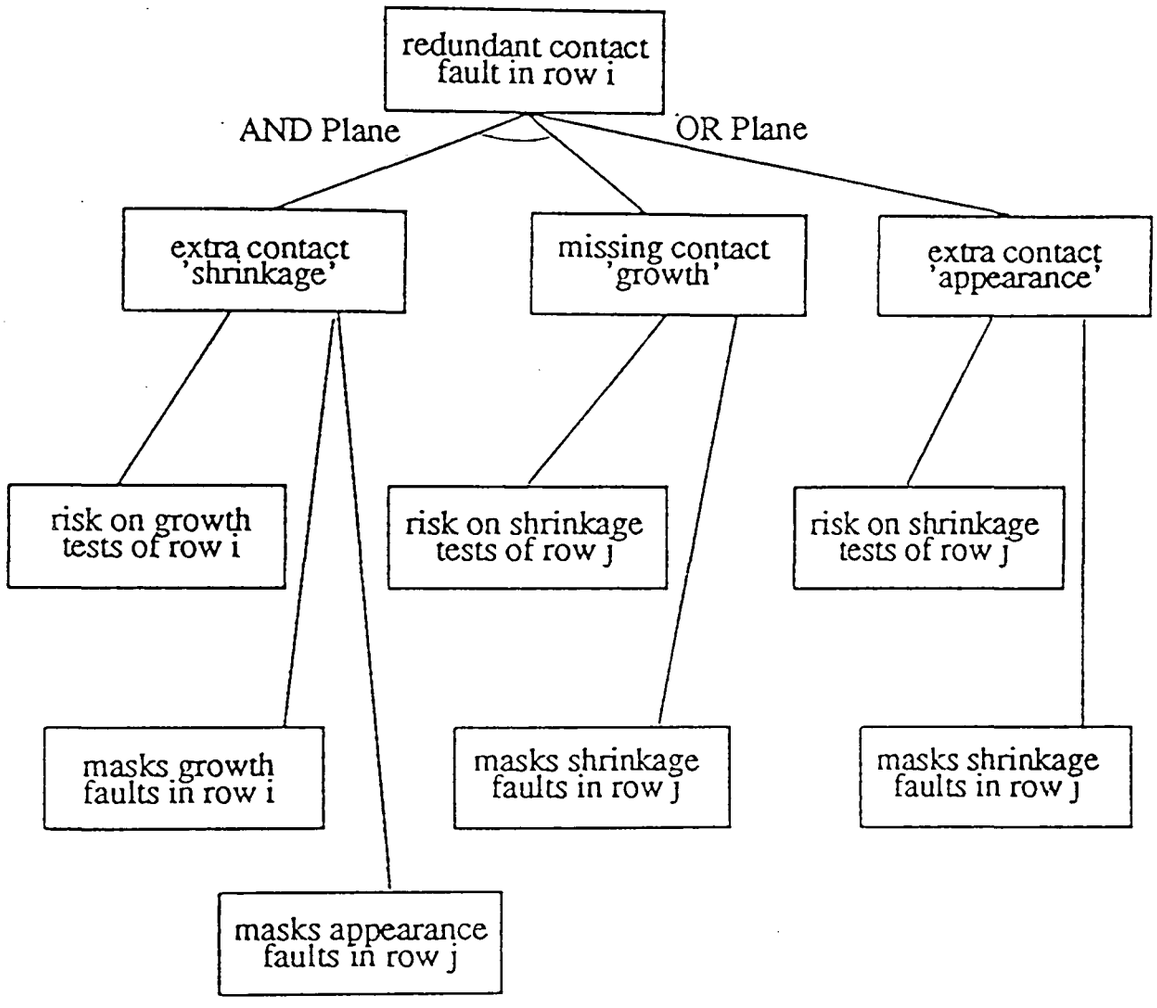


Fig. 4.1 Fault Data Structure

However, large PLAs may contain many such contacts and their existence represents anther difficulty of testing the PLA. Later on in the Chapter, a simple algorithm to identify such contacts, which cause a potential increase in the computational time of testing, shall be presented. Also, in his paper, Bose [14] has shown that most of the appearance faults could be covered by shrinkage fault test patterns. In this thesis, however, no restrictions shall be assumed on appearance tests as they represent a small percentage of the complete test set.

4.3 PROGRAM STRUCTURE

The structure of FACTPLA program is illustrated in Figure 4.2. The flow of information among the program main routines is well understood by considering each routine individually :

(1) The INPUT FILE

The file input to the program contains the description of the PLA distributed among three sets of data :

(a) the set [m,n,z,max] where

m : is the number of product terms,

n : is the number of input variables,

z : is the number of output functions, and

max : is the number of product terms belonging to the largest function; the function with the maximum number of terms.

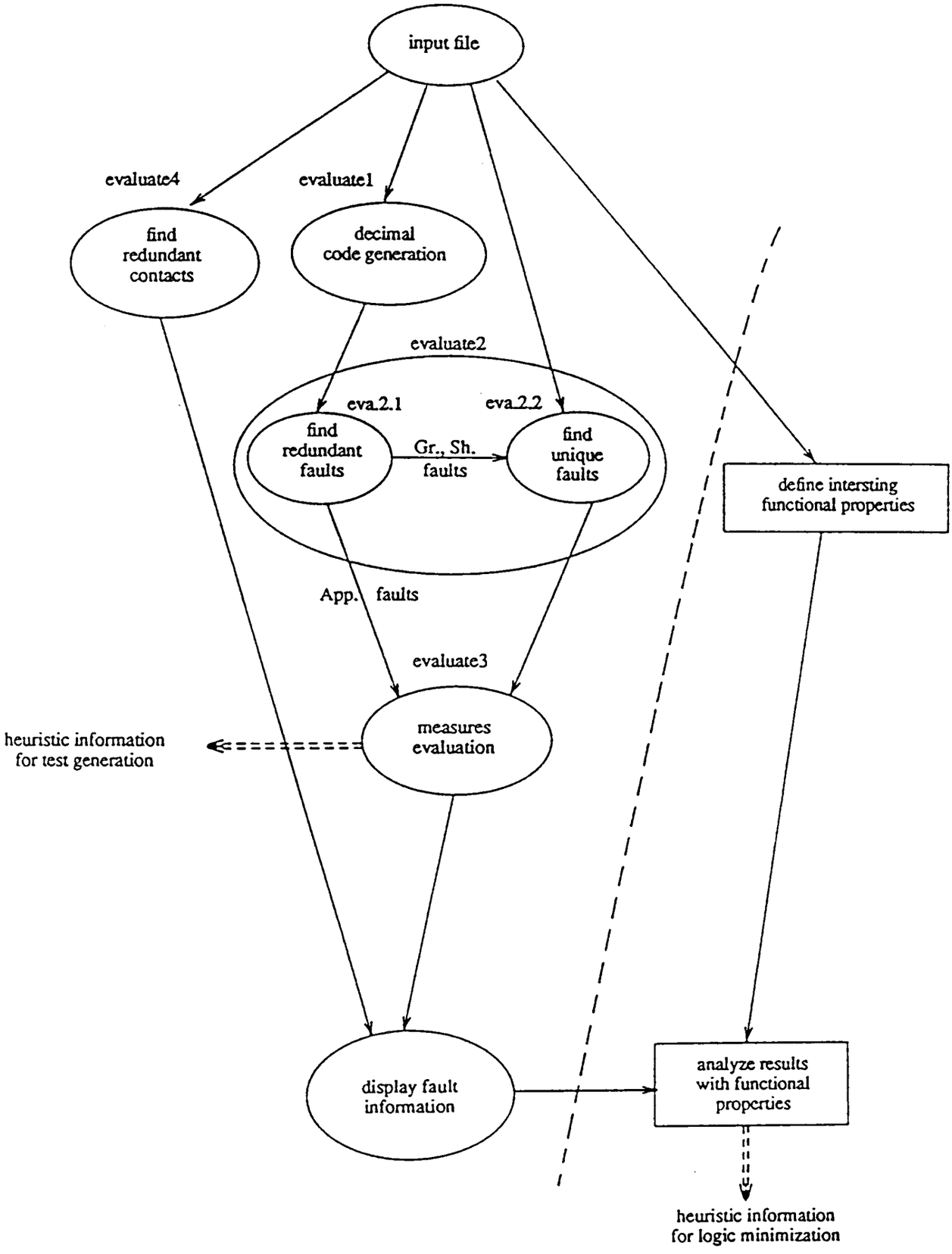


Fig. 4.2 FACTPLA Program Structure

(b) the set of product term cubes constituting the PLA.

(c) the set of output functions containing the 'numbers' of the terms in each function.

The PLA example given in Figure 3.9(a) of the previous Chapter is described to FACTPLA program as follows :

```
set (a)     6,4,3,4

            1 X X 1
            1 X 1 X
set (b)     X 1 1 1
            0 0 X X
            X 1 1 X
            X 0 0 X

            0,1,2
set (c)     3,4
            0,3,4,5
```

(2) EVALUATE1: Decimal code generation routine

The underlying heuristic in FACTPLA program is the adoption of decimal codes to represent the functionality of the PLA. The product terms are analyzed by the program as 'ordered' sets of integer numbers representing the minterms. Such arrangement has reduced the computation time and the complexity of the whole program. Figure 4.3 illustrates a general flow chart for this routine, while Appendix B contains the detailed symbolic representation describing the derivation of the decimal codes in the manner needed by the program.

(3) EVALUATE2: Fault identification routine

Having generated the decimal codes of the product terms, it is necessary to identify the set of minterms which are common (bounded) between two or more terms. Generating the

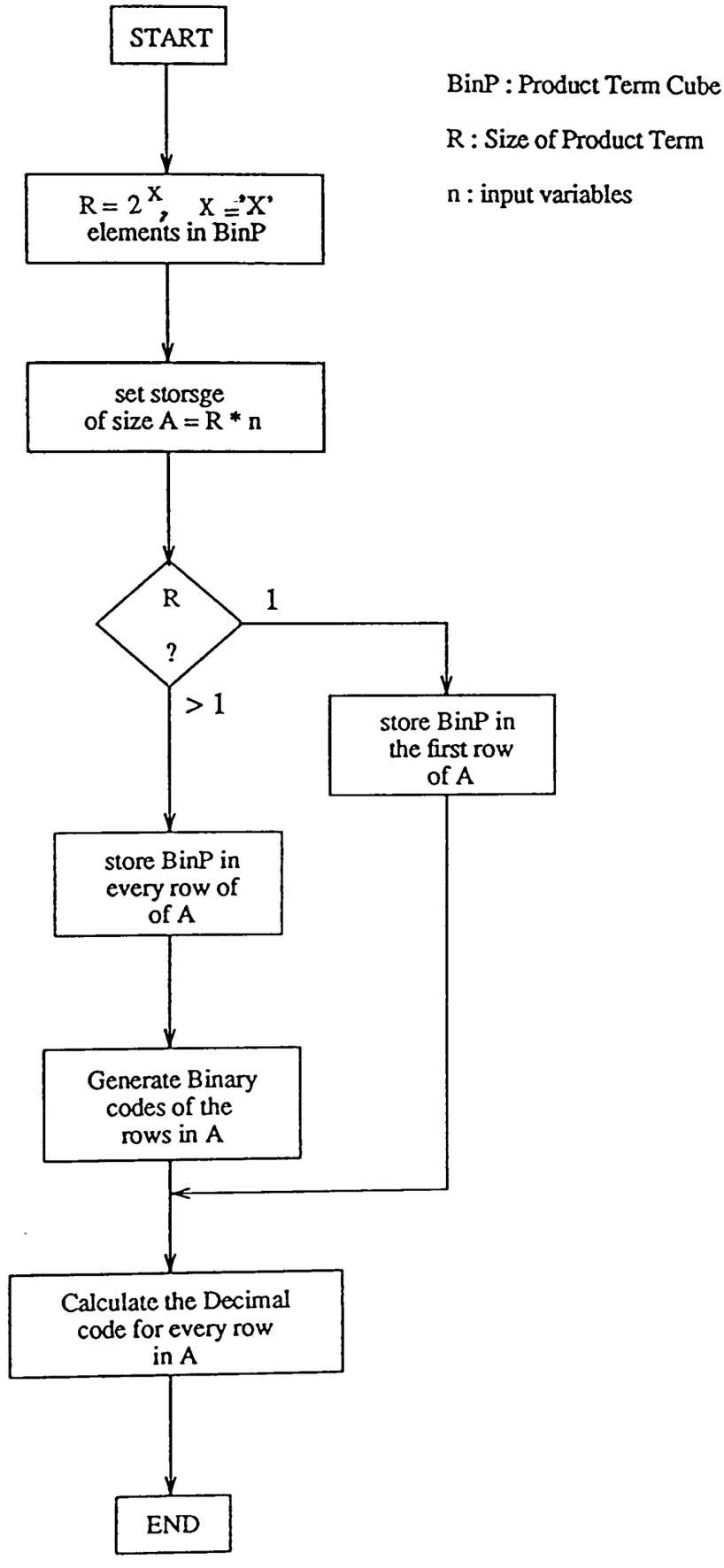


Fig. 4.3 Decimal Code Generation of Product Terms

´intersection vector´ between the terms represents the first task of this routine (see Appendix B for this step). The second and basic task of the routine is the actual identification of the redundant contact faults (growth, shrinkage, and appearance) in the PLA structure. In the next section, the detailed procedures for this step are presented. It should be noticed that FACTPLA assumes a ´complete´ single fault test set, that is, a test pattern for every testable fault is included in the test set. Therefore, a fault node is considered if and only if the corresponding fault is redundant with respect to all the relevant output functions. In other words, a fault on row i of the PLA is redundant if and only if is redundant with respect to every function containing P_i.

(4) EVALUATE3: Measures evaluation routine

This routine constitutes the heart of FACTPLA program. It evaluates the (MASK) and (RISK) measures for every possible redundant contact fault in the PLA. A special compact nodal structure has been set up to contain the fault´s information. There is a node for every class of the redundant faults, therefore, a total of 3 nodes are used for redundant growth, shrinkage, and appearance faults. Every node contains the location and the relevant difficulty measures for some redundant fault. the layout of such structure is illustrated in (6) below, while the establishment of each fault node is described in detail in the next section.

It is worth noting that the strategy of evaluating the MASK_RISK measures has also indicated the way with which good tests for multiple faults may by chosen. Such worthy information provide a good heuristic for any test generation procedure.

(5) EVALUATE4: Redundant contacts identification routine

FACTPLA program strategy assumes that a large PLA may contain some redundant contacts (devices) in the OR plane of the array. This is due to the fact that current minimization procedures for large boolean expressions may optimize some of the minimality criteria. In other words, although a PLA may be designed to contain a minimum number of product terms (or rows), some of these terms may still be redundant with respect to some of the output functions (see Fig. 3.3 in the previous Chapter). This routine evaluates the amount of these contacts which are caused by such optimization techniques.

(6) The DISPLAY routine :

This routine displays the output data in the manner presented by the EVALUATE3 routine above. The fault node is described below where

m_i : row i in the PLA,

n_j : input variable j to the PLA,

f_k : output k from the PLA, and

$$c = \begin{cases} 1 : \text{the uncomplemented column (bit line) of input } n_j \\ 0 : \text{the complemented column (bit line) of input } n_j \end{cases}$$

fault node	location	measures
growth	(m_i, n_j)	MASK, RISK
shrinkage	(m_i, n_j, c)	MASK, RISK
appearance	(m_i, f_k)	MASK, RISK

Fault Data Structure

In an advanced stage, the displayed fault information may be analyzed against the basic functional properties of the PLA prior to develop some heuristic guide information that help logic minimization techniques to arrive at the best_to_test circuit.

In the following section, FACTPLA program for single output PLAs is presented. Generalization to multiple output PLAs will be considered later on in the Chapter.

4.4 FACTPLA FOR A SIMPLE (n,m,1) PLA

In this section, the basic PLA structure is assumed to have n input variables, m product terms and one output function. Accordingly, the total number of contacts in a (n,m,1)_PLA is m(2n+1). All product lines will contribute to the output function and appearance faults need not be considered. Thus, the masking measure (MASK) is assumed to consist of two parts : the masking of detectable growth faults by some redundant shrinkage fault on the same row, and the masking of detectable shrinkage faults by some redundant growth fault on different rows of the PLA.

The restrictions on single growth fault tests for a product term P_i can be evaluated directly from the adjacent Table of P_i. Such restrictions are imposed by the existence of some redundant shrinkage fault in row i. On the other hand, since the shrinkage tests for P_i are involved in P_i itself then the restrictions on single shrinkage fault tests for P_i are evaluated from the partitions of p_i. In this case, the restrictions on shrinkage tests are imposed by the existence of some redundant growth fault in row j, $j \neq i$. Higher values of the (RISK) measure indicate that maximizing the multiple fault coverage may by achieved with longer test lengths, while higher values of the (MASK) measure reveal poor multiple fault coverage by a single fault test set.

4.4.1 Algorithms for a (n,m,1) PLA

Basically, two main analytic algorithms for redundant faults consideration in a general (n,m,1)_PLA is considered. This is illustrated by the simple flow chart given in Figure 4.4. In this Figure, algorithm 4.4.1 identifies all redundant growth faults in the product term P_i and evaluates their masking effects on the detectable shrinkage faults of other product terms. Algorithm 4.4.2 identifies all redundant shrinkage faults in P_i and evaluates their masking effects on the detectable growth faults of P_i itself. These algorithms are described below where

$(MASK)_{s\_g}$: denotes the masking of growth faults due to redundant shrinkage faults,

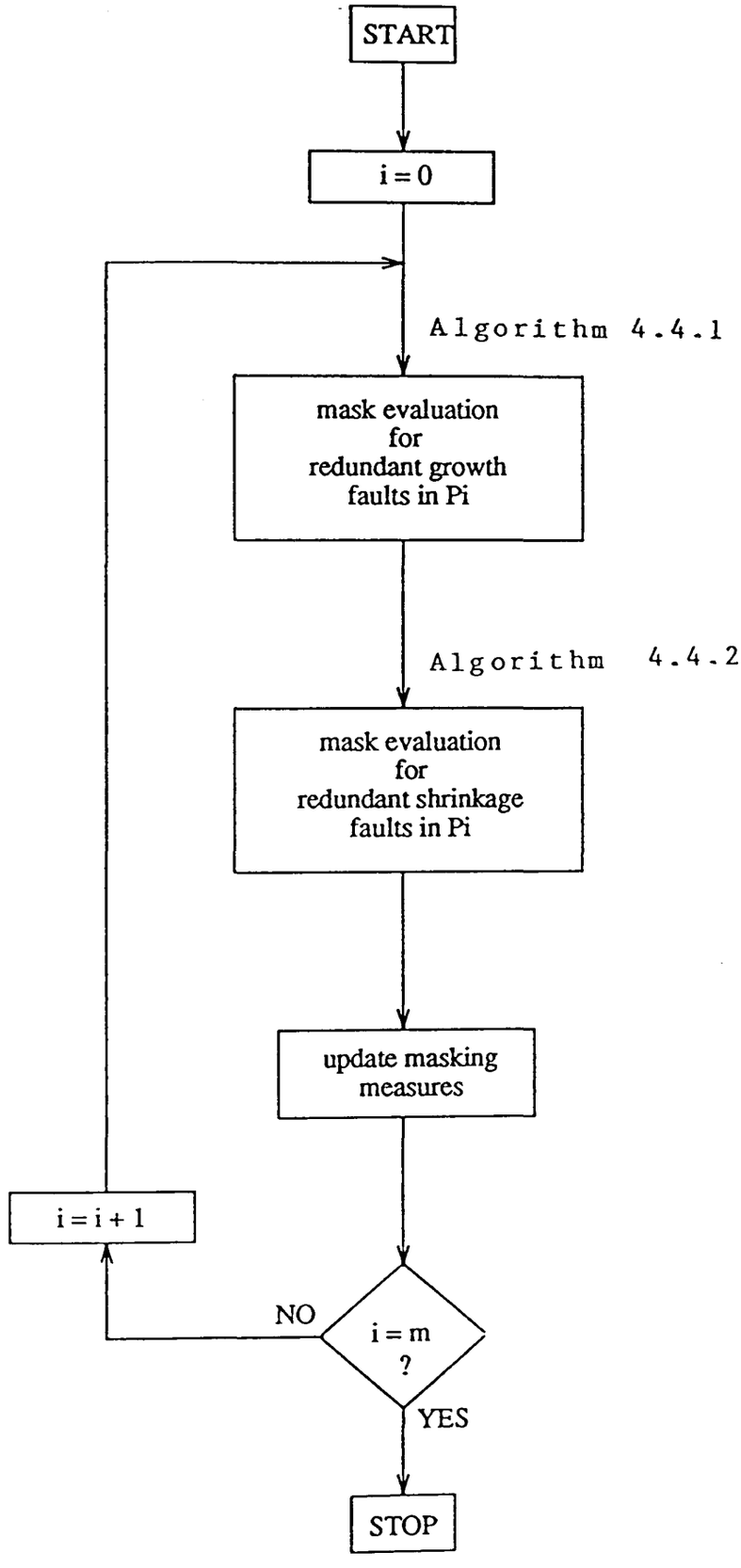


Fig. 4.4 Fault Masking Evaluation in A (n,m,1)_PLA

$(MASK)_{g\_s}$: denotes the masking of shrinkage faults due to redundant growth faults,

$(RISK)_{s\_g}$: denotes the restriction (due to redundant shrinkage faults) on growth tests to cover multiple faults,

$(RISK)_{g\_s}$: denotes the restriction (due to redundant growth faults) on shrinkage tests to cover multiple faults, and

R_i: denotes the size of P_i (number of minterms in P_i).

Again, it should be emphasized that the minterms are arranged in ascending order such that the representation of any product term begins with the lowest minterm and ends with the highest one (see the definition of $(A)_p$ in the previous chapter).

Algorithm 4.4.1 (for redundant growth faults in P_i)

Procedure 4.4.1(a) : fault identification.

Procedure 4.4.1(b) : fault location & measures evaluation.

Procedure 4.4.1(a)

1. Generate the adjacent Table of P_i as follows

(a) for each minterm (M) of P_i, obtain the set S of all possible minterms adjacent to M using the following expression :

$$S = M + [2^{(k-1)}][(-1)^{M/2^{(k-1)}}]$$

where $1 \leqslant k \leqslant n$ and $(M/2^{(k-1)})$ is defined as an integer divide.

(b) the result of (a) above is a Table of n columns and R_i rows. Element every column which is exactly identical to P_i.

2. For the Table generated in step 1, tag all the minterms belonging to the function under consideration.

3. In the above Table, any column, say $(column)_j$, whose minterms were all tagged represents a redundant column (or undetectable fault).

Procedure 4.4.1(b)

1. Set $(RISK)_{g\_s}$ and $(MASK)_{g\_s}$ to zero.

2. locate the redundant growth faults on row i in the following manner. If $(column)_j$ of the adjacent Table of P_i was redundant, then the bit change at the $(n-j)^{th}$ position of the cube of P_i is redundant.

3. evaluate $(RISK)_{g\_s}$ and $(MASK)_{g\_s}$ for $(column)_j$ as follows :

   (a) for every product term P_q, $q \neq i$, if the size $R_q = 1$ then check if the only minterm of P_q belongs to $(column)_j$. If it does, then
   $(MASK)_{g\_s} = (MASK)_{g\_s} + 1$.
   However, if the size $R_q > 1$, then go to step (b) below.

   (b) perform the partitioning process on P_q. Every partition of P_q will have two blocks, and every block

contains an implicant of size $R_q/2$ (Appendix B illustrates a symbolic representation for such partitioning process). Tag the bounded minterms in the partition blocks. For any block, say $(block)_k$, if the set of 'unique' minterms in $(block)_k$ were all belonged to $(column)_j$ then

$(MASK)_{g\_s} = (MASK)_{g\_s} + 1.$

Otherwise, for every unique minterm, in $(block)_k$, that is belonged to $(column)_j$ do

$(RISK)_{g\_s} = (RISK)_{g\_s} + 1.$

4. Repeat step 2 and 3 for every redundant column in the table generated by procedure 4.4.1(a).

Figure 4.5 illustrates the application of algorithm 4.4.1 on the first row of a simple PLA.

Algorithm 4.4.2 (for redundant shrinkage faults in P_i)

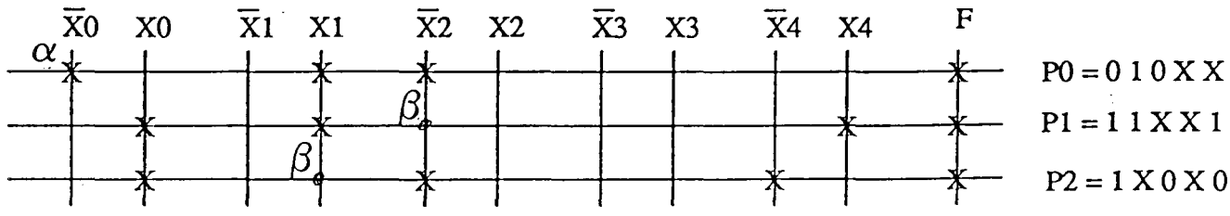
Procedure 4.4.2(a) : fault identification & location

Procedure 4.4.2(b) : measures evaluation

Procedure 4.4.2(a)

1. Perform the partitioning process on P_i to obtain the $\log_2 R_i$ possible partitions and tag the bounded minterms.

2. Scan partitions of P_i. If, for a given partition, say $(PT)_j = [(block)_a, (block)_b]$, the set of the minterms belonging to one block, say $(block)_a$, were all tagged (redundant block), then mark this block as a redundant fault in row i. Obviously, the other block, $(block)_b$ in this case, represents a shrunk



Step 1 :

	c0	c1	c2	c3	c4
8	9	10	12	0	24
9	8	11	13	1	25
10	11	8	14	2	26
11	10	9	15	3	27

(a)

8	-	-	12	0	24
9	-	-	13	1	25
10	-	-	14	2	26
11	-	-	15	3	27

(b)

Step 2 & Step 3

8	-	-	12	0	24/
9	-	-	13	1	25/
10	-	-	14	2	26/
11	-	-	15	3	27/

↑ redundant

Procedure 4.4.1(a)

Step 1 : RISK = MASK = 0

Step 2 : fault location = n - c4 = 0 (first bit change 0 → X)

Step 3 :

masked ↙

[(25 , 27) (29 , 31)]

Partitions of P1

[(25 , 29) (27 , 31)]

masked ↙

[(16 , 18) (24 , 26)]

Partitions of P2

[(16 , 24) (18 , 26)]

Procedure 4.4.1(b)

Fig. 4.5 Application of algorithm 4.4.1 on P0

term caused by this particular fault. The location of the fault is determined from the order of the partitions in the manner described in the previous Chapter.

Procedure 4.4.2(b)

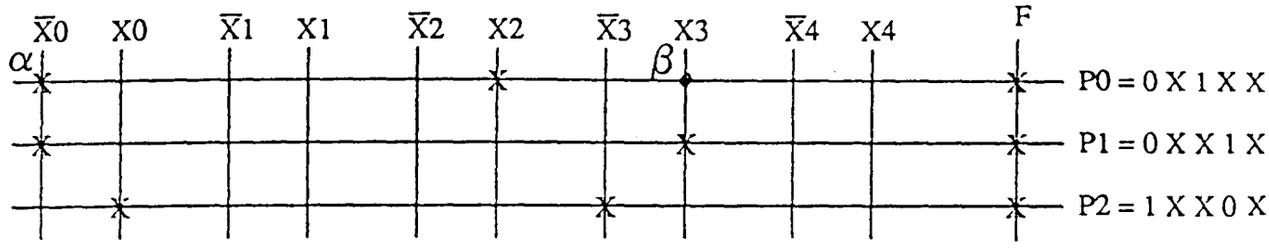
1. Set $(RISK)_{s\_g}$ and $(MASK)_{s\_g}$ to zero.
2. Assume that a block, say $(block)_a$, in partition j of P_i was redundant. Then obtain the 'adjacent Table' of the other block, $(block)_b$, and perform the necessary tagging (note that the whole original product term P_i should be removed from the Table). In this Table, for every column whose minterms were all tagged do
   $(MASK)_{s\_g} = (MASK)_{s\_g} + 1.$
   Otherwise, for every tagged minterm (in the above column) do
   $(RISK)_{s\_g} = (RISK)_{s\_g} + 1;$
3. Repeat step 2 for every partition of P_i.

Figure 4.6 illustrates the application of algorithm 4.4.2 on the first row of a simple PLA.

4.4.2 Application on Switching Theory

Most minimization procedures tend to obtain a minimal sum_of_products expression for a given switching function, after establishing some criteria for minimality. Consider the minimization of the function F(x,y,z) :

$F(x,y,z) = \bar{x}y\bar{z} + \bar{x}\bar{y}\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + xyz + x\bar{y}z$



partitions of P0

Step 1 & Step 2

[(4 , 5 , 6*, 7*) (12 , 13 , 14*, 15*)] redundant block

[(4 , 5 , 12 , 13) (6*, 7*, 14*, 15*)]

[(4 , 6*, 12 , 14*) (5 , 7*, 13 , 15*)]

the second X_element bit change (X --> 1) is undetectable

Procedure 4.4.2(a)

Step 1 : RISK = MASK = 0

Step 2 : obtain the adjacent Table of shrunk term

4	5	6	0	20
5	4	7	1	21
12	13	14	8	28
13	12	15	9	29

the original p_term

4	-	-	0	20 /
5	-	-	1	21 /
12	-	-	8	28 /
13	-	-	9	29 /

masked

Procedure 4.4.2(b)

Fig. 4.6 Application of algorithm 4.4.2 on P0

Combining the first and second, second and third, forth and fifth, fifth and sixth terms yields a reduced function for F :

$F(x,y,z) = \bar{x}\bar{z} + \bar{y}\bar{z} + yz + xz$ ..... (1)

The above sum of products expression is said to be irredundant or irreducible, since no term or literal can be deleted without altering its logical value. However, combining the first and second, third and sixth, forth and fifth terms of F results in :

$F(x,y,z) = \bar{x}\bar{z} + x\bar{y} + yz$ ..... (2)

Similarly, the combinations of the first and forth, second and third, fifth and sixth terms yield a third irredundant expression :

$F(x,y,z) = \bar{x}y + \bar{y}\bar{z} + xz$ ..... (3)

While all three expressions are irredundant, only the latter two are minimal. Consequently, an irredundant expression is not necessarily minimal, nor is the minimal expression always unique. Note that the minimality criteria depend on two parameters : the number of prime implicants and the number of literals in each such prime implicant.

Now, realized as PLA structures, expression 2 and 3 yields the same number of product lines and contacts (connections) in the array. Thus, from the design point of view, the choice between the two possible realizations seems to be arbitrary. However, the algorithms presented

ection assume a third (testability)

criterion for chosing the best PLA realization. For example, Figure 4.7 shows 3 different single output PLA structures which realize the same switching function F. The PLAs are specified by structural personalities where they contain the same number of contacts (devices). Note that all the structures have the same number of undetectable single contact faults. Nevertheless, the first PLA is shown to have less difficulty measures and, hence, the easiest to test structure. In the concluding part of this thesis, the general functional properties that help arriving at the 'best' realization are discussed.

4.5 FACTPLA GENERALIZATION TO MULTIPLE OUTPUT PLAs

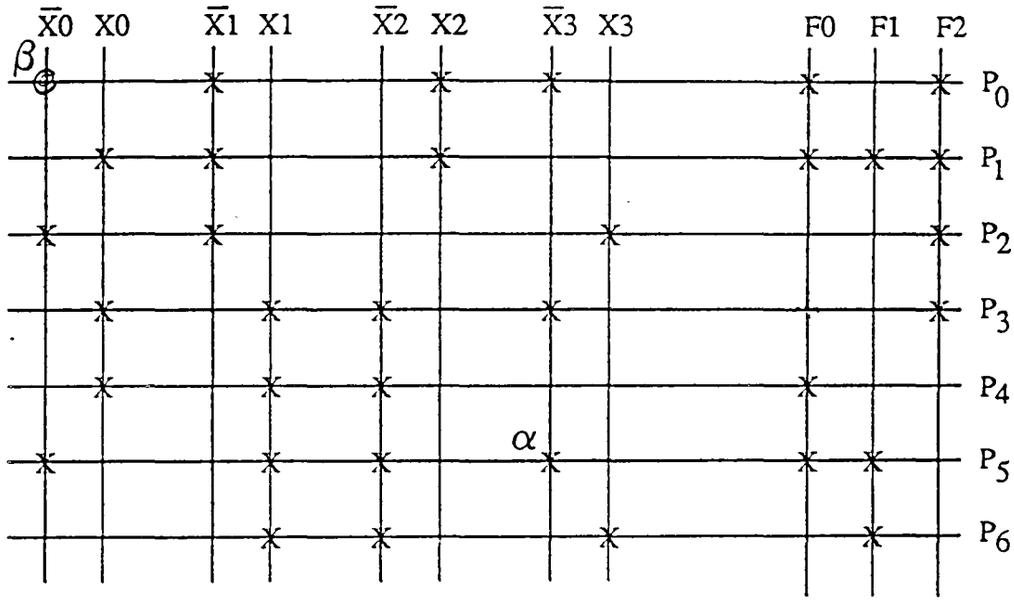
In the case of multiple output PLAs, shrinkage and growth faults in any row must be checked through different output functions since a complete single fault test set is assumed to exist always. A fault q, where q belongs to the union set of all shrinkage and growth faults in row i, is redundant if and only if q is redundant with respect to all output functions containing P_i. In Figure 4.8 , the missing contact fault α causes P_5 to grow. Now, although α is redundant with respect to the output function f_1, it can still be detected through f_0. On the other hand, the shrinkage fault β of the product term P_0 can not be detected through f_0 or f_2. Thus, β is redundant and it must be considered for fault masking evaluation.

F	000	001	011	010	110	111	101	100
00	0	4 1	12 1	8	24	28	20 1	16
01	1 1	5 1	13 1	9	25 1	29	21 1	17
11	3 1	7 1	15 1	11 1	27 1	31	23 1	19 1
10	2	6 1	14 1	10 1	26	30	22 1	18 1

	PLA_1	PLA_2	PLA_3
p0	x 0 1 x x	x 0 1 x x	x 0 1 x x
p1	0 x 1 x x	0 x 1 x x	0 x 1 x x
p2	0 0 x x 1	0 0 x x 1	0 0 x x 1
p3	1 1 0 x 1	1 1 0 x 1	1 1 0 x 1
p4	1 x 0 1 x	1 x 0 1 x	1 0 x 1 x
p5	0 1 x 1 x	x 1 0 1 x	x 1 0 1 x

	undetectable single faults	(mask) measure	(risk) measure
PLA_1	6	1	8
PLA_2	6	2	11
PLA_3	6	1	10

Fig. 4.7 Testability Measures for Different PLA Realizations



fault α : redundant with respect to F1

testable with respect to F0

fault β : redundant with respect to F0 & F2

Fig. 4.8 Fault Detection in Multiple Output PLAs

In addition to the above faults, appearance faults must now be considered. As it was pointed out in the previous Chapter, identifying appearance faults and evaluating their masking influences are based on two objects; a product term P_i and an output function f_j such that P_i does not belong to f_j.

Also, In VLSI environments logic functions may have up to 30 input and output and more than 100 product terms. Thus, exact logic minimization, which involve generation of all prime implicants and extraction of a minimum prime cover, is impractical. indeed, the problem of extracting a minimum prime cover is known to belong to the class of NP_complete problems[38]. Obviously, the computation time of such problems increases drastically with the increase number of implicants. Hence, the need for optimization techniques to generate a near minimum cover may results in the inclusion of some redundant contacts in the OR plane of a PLA.

Accordingly, the following algorithms complete the fault masking analysis for a general PLA structure. Algorithm 4.5.1 evaluates redundant shrinkage and appearance fault effects in row i of the PLA, while algorithm 4.5.2 identifies all redundant contacts in the OR plane of the array.

Algorithm 4.5.1 (for a product term P_i)

In this algorithm let

$(MASK)_{a\_s}$: denotes the masking of shrinkage faults due to redundant appearance faults of P_i,

$(MASK)_{s\_a}$: denotes the masking of appearance faults due to redundant shrinkage faults in P_i,

$(RISK)_{a\_s}$: denotes the restriction (due to redundant appearance faults) on shrinkage tests to cover multiple faults.

Procedure 4.5.1(a) : redundant shrinkage faults effects

Procedure 4.5.1(b) : redundant appearance faults effects

Procedure 4.5.1(a) (on the same row of the PLA)

1. Set $(MASK)_{s\_a}$ to zero.
2. Assume that $(block)_a$ of partition j of the product term P_i was redundant (see algorithm 4.2.2). Thus the other block, $(block)_b$ in this case, represents the shrunk term of P_i. For every output function, f_k, such that P_i does not belong to f_k and $(block)_b$ belongs to the set of terms constituting f_k do
   $(MASK)_{s\_a} = (MASK)_{s\_a} + 1$.
3. Repeat step 2 for the other redundant blocks in P_i´s partitions.

Procedure 4.5.1(b) (on different rows of the PLA)

1. Set $(RISK)_{a\_s}$ and $(MASK)_{a\_s}$ to zero.
2. For every output function, f_k, such that P_i does not belong to f_k do
   (a) if P_i belongs to the ´set of terms´ constituting f_k, then mark the extra contact fault at the junction between P_i and f_k as a redundant appearance fault.
   (b) for every product term P_j, $j \neq i$ and P_j belongs to

f_k, perform the partitioning process to obtain P_j's partitions; $(PT)_{Pj}$. Any block of $(PT)_{Pj}$ whose 'unique' (with respect to f_k) minterms were all belonged to P_i represents a masking condition. Thus, for every such condition do

$(MASK)_{a\_s} = (MASK)_{a\_s} + 1.$

Otherwise, for every minterm (in the above block) which is unique with respect to f_k and belong to P_i do

$(RISK)_{a\_s} = (RISK)_{a\_s} + 1.$

(c) repeat (b) for every block in the set of P_j's partitions.

Algorithm 4.5.2 (Identification of redundant contacts)

Let R_dev be denotes the total number of redundant contacts in the OR plane of the PLA.

1. Set R_dev to ZERO.
2. For an output function f_i, any product term P_j belonging to f_i such that all of P_j's minterms are also 'bounded' with respect to f_i then

   R_dev = R_dev + 1.
3. Repeat step 2 for every output function.

4.6 EXPERIMENTAL RESULTS

In this section, the experience of applying FACTPLA on different PLA structures is discussed. Appendix C contains data on 13 different PLAs, which have been

collected from various sources, while Table 4.1 summarizes the results given in the Appendix according to FACTPLA.

The first column of Table 4.1 contains the PLAs arranged according to their alphanumeric names. The amount of undetectable single contact faults are given in the second column. These faults are redundant with respect to any functionally generated single fault test test.

The RISK and MASK values, given in the third and forth columns respectively, reveal the impact of 'bad' design on multiple faults coverage in a PLA. The last column in Table 4.1 gives an idea about the amount of undetectable faults as compared to the total number of possible growth, shrinkage, and appearance faults, that is, for a (n,m,f)_PLA :

$$\text{undetectability \%} = \frac{\text{total redundant faults}}{\text{Total possible faults}} * 100$$

where

$$\left.\begin{aligned}\text{Total growth faults} &= \sum_{i=1}^{m} (n - \text{Log}_2 R_i) \\ \text{Total shrinkage faults} &= \sum_{i=1}^{m} (2 * \text{Log}_2 R_i)\end{aligned}\right\} \text{see property 3.3.1}$$

and

$$\left.\text{Total appearance faults} = (m * f) - \Big(\sum_{i=1}^{m} \sum_{(j=1)}^{f} j \Big)_{p_i \notin f_j} \right\} \text{see section 3.4.3}$$

Now, it is very convenient to normalize the RISK and MASK values by certain parameters so that the complexity of testing a PLA may be estimated according to the personality structure of the PLA itself. Before proceeding to do so, the theoretical upper bound on the test length for a PLA has to be defined.

PLA	single undetectable faults	Total RISK	Total MASK	undetectability %
PLA_5X	157	224	305	11%
PLA_BW	1148	43	1075	42%
PLA_BW1	496	51	511	23%
PLA_CON1	7	38	3	6%
PLA_DIL	15	1168	10	3%
PLA_F2	12	1	12	12%
PLA_MAS	42	65	140	5%
PLA_MID	1	0	40	0.37%
PLA_MISEX1	99	220	499	17%
PLA_RD53	20	4	10	8%
PLA_RD73	199	34	73	14%
PLA_SAO2	111	552	104	12%
PLA_SR	16	180	0	6%

Table 4.1 Testability Profile for differents PLAs (I)

Since the strategy of FACTPLA program assumes that growth and shrinkage tests constitute the vast majority of the patterns in the test length, only these tests shall be considered henceforth. Now, in the previous chapter it has been shown that up to 3 patterns may be needed to detect all testable shrinkage faults in a general product term (see section 3.3). Therefore, in a PLA with m rows the upper bound on the shrinkage tests is 3m. On the other hand, a product term grows into different coordinates and no two growths can overlap. Hence, the worst case for testing growth faults is to have a distinct test pattern for every possible missing contact fault, that is ($n - \mathrm{Log}_2 R$).

It follows immediately that the theoretical upper bound on the test length for a (n,m,f)_PLA may be given by

$$\text{Theoretical upper bound on test length} = 3 * m + \sum_{i=1}^{m} (n - \mathrm{Log}_2 R_i)$$

Accordingly, the worst case for covering multiple faults in a PLA is to assume that all the patterns having RISK values do belong to the test set. Therefore, the testability criterion of multiple faults in a PLA may be given by

$$\text{RISK on Multiple fault coverage \%} = \frac{\text{Total RISK}}{\text{Theoretical upper bound on the test length}} * 100$$

Similarly, since the MASK values account for those testable faults which are masked from detection, then it is very convenient to normalize these values by the amount of the testable faults, i.e.,

$$\text{MASK \%} = \frac{\text{Total MASK}}{\text{Total testable faults}} * 100$$

where

testable faults = Total faults - Total redundant faults

The normalized values for the PLA examples in question are summarized in Table 4.2. Note that some of the RISK and MASK measures in Table 4.2 have an effect of (>100%). This is explained below.

The FACTPLA program is a pre_test generation technique which tends to estimate the complexity of testing a PLA. Hence, the values of RISK measures are evaluated for all possible qualified test patterns without knowing exactly which tests will be included in the final test length. Therefore, without loss of generality, it is natural to assume that, at the worst case, all tests with RISK values are included in the final test length. In some cases, the amount of such tests may exceeds the theoretical upper bound of the test length.

On the other hand, a testable contact fault may be masked by more than one redundant fault under various tests. For instance, a testable growth fault on row i of a PLA may be masked by more than one shrinkage fault on the same row.

PLA	theoretical upper bound on test length	Single detectable faults	RISK %	MASK %
PLA_5X	512	1272	43%	24%
PLA_BW	611	1557	7%	57%
PLA_BW1	435	1620	11%	31%
PLA_CON1	50	105	76%	2%
PLA_DIL	186	437	>100%	2%
PLA_F2	72	84	1%	14%
PLA_MAS	535	712	12%	19%
PLA_MID	97	270	0%	15%
PLA_MISEX1	218	483	>100%	>100%
PLA_RD53	240	218	1%	4%
PLA_RD73	1263	1217	2%	6%
PLA_SAO2	597	780	92%	13%
PLA_SR	139	220	>100%	0%

Table 4.2 Testability Profile for different PLAs (II)

PLA	Complexity of testing
PLA_MISEX1	>100%
PLA_SAO2	52%
PLA_DIL	51%
PLA_SR	50%
PLA_CON1	39%
PLA_5X	33%
PLA_BW	32%
PLA_BW1	21%
PLA_MAS	15%
PLA_MID	7%
PLA_F2	7%
PLA_RD73	4%
PLA_RD53	2%

Table 4.3 The Complexity of Testing different PLAs (III)

Hence, the amount of masking conditions may exceeds the total number of single testable faults in the PLA.

However, for more a accurate estimation of the complexity of testing a PLA, both measures have to be considered simultaneously. In this case, the complexity of testing indicates the effectiveness of a single fault test set to cover more multiple faults. Such complexity will be defined by the mean value of both measures :

$$\text{complexity of testing } \% = \frac{\text{RISK\%} + \text{MASK\%}}{2}$$

In Table 4.3 the PLA examples in question are re_arranged according to their testing complexity defined by the above expression.

4.7 SUMMARY

The analytic program presented in the previous sections produces measures for testability investigation in PLAs. The procedures of the FACTPLA program may be considered for estimating the complexity of the whole program. For instance, the memory requirement for generating and manipulating the adjacent tables and partitions of the product terms is limited by the size of the largest product term. At any instant of the program, only ´one´ adjacent Table and ´one set´ of partitions are required.

Accordingly, the storage required in the main computation of the FACTPLA program may be described below

$$\underbrace{\text{Log}_2 R_{max} \cdot R_{max}}_{\text{for partitions}} + \underbrace{R_{max} \cdot n}_{\text{for adjacent tables}}$$

where $\text{Log}_2 R_{max}$: is the number of partitions in the largest product term,

R_{max} : is the size of the largest product term, and

n : is the number of inputs to the PLA.

It follows that the operations in FACTPLA relate linearly to the number of product terms and the amount of redundant faults. Both MASK and RISK evaluations are completed after a single pass through the set of product terms.

Another factor to consider here is the ease with which 'more realistic' test patterns can be generated for a PLA. The functional verification approach embodied in the program may be used to obtain such patterns, that is, the PLA under normal condition performs the intended operation, even if some redundant faults exist. It is obvious that the derivation of such patterns can be achieved directly from the adjacent Tables and partitions of the product terms. A straightforward application of any minimal cover routine on these Tables and partitions yields a minimal test length for the PLA.

Now, to achieve minimal fault masking and higher coverage of multiple faults, the adjacent Tables and partitions may be updated by removing those 'free' minterms (from the Tables) and 'unique' minterms (from the partitions) which

belong to redundant fault sets (columns in the Tables and blocks in the partitions). Thus, applying the same minimal cover routine, another test length can be achieved and a new testability criterion may be established by comparing both test lengths. It is worth noting that increasing the fault coverage is achieved without augmenting the structure of the PLA, therefore, reducing the chances of fault occurrence in the sequential circuits involved in most of the augmentation techniques [31-37].

However, FACTPLA program analyzes the functional specification of PLAs without considering the topology of the arrays as such. Thus, as far as the properties of the output function(s) remain the same, the program is technology invariant and may also be applied to the folded versions of the PLAs.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORK

5.1 CONCLUDING REMARKS

With the advent of VLSI, the circuit complexity of chips has been increasing exponentially. Considerable effort has been made to incorporate regular structures into circuit design. As one of such structures, Programmable logic arrays presently occupy an extremely useful role in the design of complex VLSI chips. On the other hand, the steadily growing chip complexity is associated with testing difficulties, particularly in the area of multiple fault detection in these chips. It is for this reason that considerable attention is being devoted for testing PLAs effectively.

The introductory part of this thesis identified the need to consider the influences of undetectable faults on testing and testability aspects in digital circuits. Being the example vehicle of the above consideration, a PLA is introduced in Chapter 2 and the problem of testing PLAs is identified in the light of the following topics :

1. Fault modeling : relates to determining how various malfunctions can be logically or functionally represented.

2. Derivation of a complete single contact fault test set : relates to the computational difficulties despite various heuristic_based test generation algorithms reported in the literature.

3. Multiple fault detection : relates to the problem of quantitatively predicting the multiple fault coverage

capability of a single fault detection test set (T_s) in a PLA. Moreover, augmenting T_s in order to obtain a multiple fault detection test set (T_m) represents another problem.

4. Effect of untestable contact faults : relates to the increasing complexity of the computations involved in the algorithms mentioned in (2) above. Furthermore, untestable contact faults become important when one attempts to show that most other faults can be covered by test patterns designed to cover only single crosspoint faults.

5. Designing easily testable PLAs : relates to the different trade_offs over the parameters associated with the extra test circuitry (e.g., the number of additional pins, fault coverage, the number of test patterns,... etc.).

The work established in this thesis presents an alternative for estimating the complexity of testing in PLAs. The described approach expresses such complexity in terms of the effects of masking among faults. Undetectable faults identification and fault masking evaluation have been combined to yield an analytic program for testability investigation in PLAs. The program (FACTPLA) analyses the functional specification and investigates the adjacency relationships among the product terms constituting the PLA.

Two testability measures, related to the effect of fault masking and restriction on single fault test patterns to cover multiple faults, are produced. A record of all undetectable contact faults is also given. The main application of the program is to distinguish between different PLA structures by considering their testability measures.

Since the properties of the output function(s) are investigated without considering any topological aspect, the program can be applied to large PLAs and to the folded PLA structures as well.

5.2 FUTURE WORK

The main aim of traditional minimization techniques is to simplify a boolean function $f(x_1,x_2,\ldots,x_n)$ to find an expression $g(x_1,x_2,\ldots,x_n)$ which is equivalent to f and which minimizes some cost criteria. The most common cost criteria are :

1. minimum number of appearances of literals (complemented or uncomplemented) in a product term,
2. minimum number of literals in a sum of products (sop) expression, and
3. minimum number of terms in a sum of products expression provided there is no other such expression with the same number of terms and with fewer literals.

Now, cost is defined as merely the number of AND (or the product terms) gates required in the realization. Such

definition is quite natural for the PLA. Thus, the third criterion above seems to be the most relevant one for PLA structures. However, as it was illustrated in chapter 4, a minimal sum of products expression produced by most minimization techniques is not necessary unique. Hence, for the same output function, there may exist several PLA realizations with different functional properties affecting the array's testability.

Now, the values calculated for the testability measures in this thesis are used to estimate the testability within a particular PLA as well as to compare the testability of different PLAs. This is due to the normalization of the absolute measures by certain common factors which are affected significantly by the functionality (realization) of each PLA. This is clearly illustrated by the histogram format (Figure 5.1) developed to show that the amount of crosspoints (as a function of the area of a PLA) is irrelevant to the complexity of testing, and hence, the testability of a PLA. Therefore, a worthy motivation towards further research may be directed to :

1. define those functional properties that affect the complexity of testing. For instance, unateness, linear separability, symmetry ..etc., may be proved to have good or bad impact on testing.

2. define the 'design style' that help reduce the complexity of testing. As far as a PLA is concerned, the design style may be confined by the 'shape' of the output (OR) plane of the array. For example, in

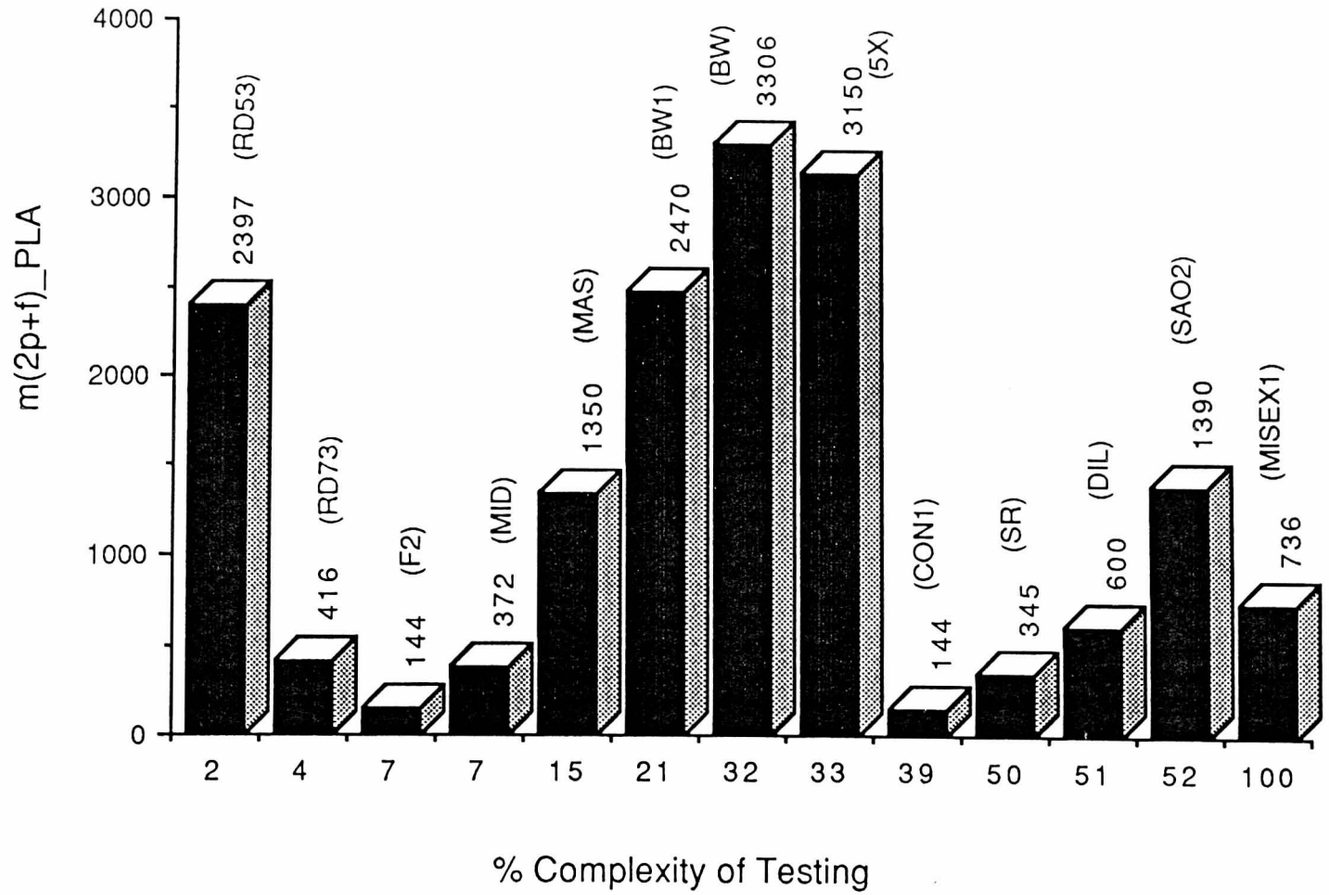
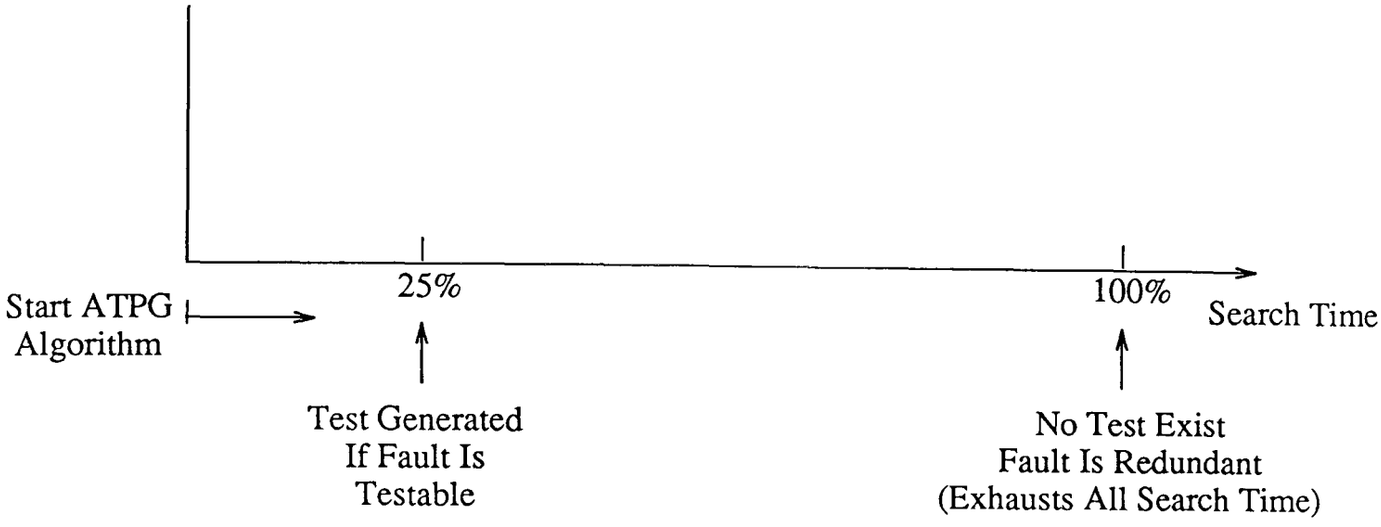


Fig 5.1 Testability Profile for Different Sizes PLAs

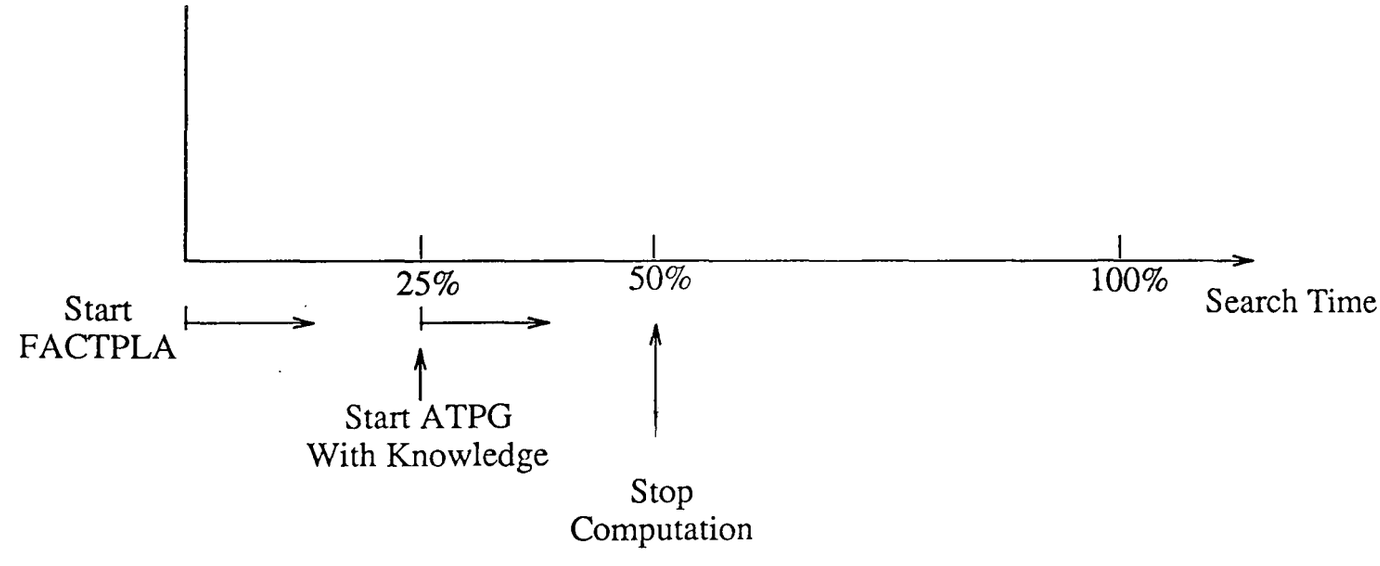
some cases a PLA is designed such that each of its product lines is belonged to one and only one output function (in Appendix C, PLA_5X, PLA_MISEX1, and PLA_RD73 are some examples of such design).

A systematic way to identify the impact of the combination of the above two parameters need to be established. Such motivation will be of great theoretical and practical interset. Furthermore, generalizing and integrating FACTPLA within a complete CAD tool for logic minimization ´with emphasis on testability´ is appreciated when considering the following argument.

Previous experience on generating tests for digital circuits shows that most of the search time of an ATPG algorithm is wasted on undetectable faults (see Figure 5.2a). Now, the knowledge provided by FACTPLA could be used to guide the ATPG algorithm if it picks up an undetectable fault. At the worst case, the computer run time of FACTPLA may be equivalent to that of the ATPG algorithm. The important conclusion of this fact is depicted in sketch (b) of Figure 5.2 where the acceleration of the ATPG algorithm is achieved. An effective reduction in the total search time is not the only advantage of FACTPLA. Another area which has been proven to be of value in Chapter four is the consideration of multiple faults. Essentially, the generation of effective test patterns which detect as many multiple faults as possible is achieved.



(a)



(b)

Fig. 5.2 Reduction In Total Search Time Of An ATPG

REFERENCES

[1] Melvin A. Breuer, Arthur D. Friedman
"Diagnosis and Reliable Design of Digital Systems".
Computer Science Press, Inc. 1976.

---

[2] P. J. Hicks
"Semi_Custom IC Design and VLSI".
Peter Peregrinus Ltd. 1983.

---

[3] T. W. Williams (Editor)
"VLSI Testing".
Elsevier Science Publishers B.V. 1986, North Holand.

---

[4] B. R. Wilkins
"Testing Digital Circuits".
van Nostrand Reinhold co. Ltd. 1986.

---

[5] R. G. Bennetts
"Design of Testable Logic Circuits".
Addison_Wesley 1984.

---

[6] H. Flleisher, L. I. Maissel
"An Introduction to Array Logic"
IBM J. Res. Dev., vol.19, March 1975.

---

[7] Thomas W. Williams , Kenneth P. Parker
"Design for Testability - A Servey".
IEEE Transc. on comp., C_31, No.1, Jan. 1982.

---

[8] J. E. Stephenson, J. Grason
"A Testability Measure for Register Transfer
Level Digital Circuits".
Dig. 6th Int. Symp. Fault-Tolerant Comput. (FTCS-6),
pp. 101-107, 1976.

---

[9] Lawrence H. Goldsteine, Evelyn L. Thigpen
"SCOAP : Sandia Controllability/Observability
Analysis Program".
IEEE 17th Design Automation (DA) conf. 1980.

---

[10] P. G. Kovijanic
"Testability Analysis".
Dig. IEEE Test Conf., pp. 310-316, Oct. 1979.

---

[11] Ion M. Ratiu, Alberto Sangiovanni_Vincentelli
"VICTOR : A Fast VLSI Testability Analysis Algorithm".
IEEE Test conf. 1982.

---

[12] R. G. Bennette, C. M. Maunder, G. D. Robinson
"CAMELOT : A Computer_Aided_Measure for Logic Testability".
IEEE ROCE, vol.128, No.5, Sep. 1981.

---

[13] I. Kohavi
"Fault Diagnosis of Logical Circuits".
Proc. 10th Ann. symp. on Switching and Automata Theory Oct. 1969.

[14] Pradip Bose, Jacob A. Abraham
"Test Pattern Generation for PLAs".
19th design Automation conf., 1982.
---
[15] Kenyon C. Y. Mei
"Bridging and Stuck_at Faults".
IEEE Transc. on comp., C_23, No.7, July 1974.
---
[16] Charles W. Cha
"A Testing Strategy for PLAs".
15th Design Automation conf., Las Vegas, June 1978.
---
[17] James E. Smith
"Detection of Faults in PLAs".
IEEE transc. on comp. C_28, No.11, Nov. 1979
---
[18] Fabio Somenzi, Silvano Gai, Marco Mezzalama, Paolo Prinetto
"A new Integrated System for PLA testing and Verification".
20th Design Automation conf. 1983.
---
[19] E. I. Muehldnof, T. W. Williams
"Optimized Stuck Fault Test Pattern Generation for PLA Macros".
Semiconductor Test symp., Oct. 1977.
---
[20] C. W. Cha
"Prime Faults in A Double_Bit Partition PLAs".
IBM Tech. Disclosure Bulletin, vol.18, No.8, 1976.
---
[21] C. W. Cha
"Test Pattern Generation for Shorts in PLAs".
IBM Tech. Disclosure Bulletin, vol.18, No.5, 1975.
---
[22] Daniel L. Ostapko, Se June Hong
"Fault Analysis and Test Generation for PLAs".
IEEE transc. on comp. C_28, No.9, Sept. 1979.
---
[23] E.B. Eichelberger, E. Lindbloom
"A Heuristic Test Pattern Generation for PLAs".
IBM Res. Dev., vol.24, No.1, Jan. 1980.
---
[24] William I. Fletcher
"Engineering Approach to Digital Design".
Prentice/Hall Int. Edition 1980.
---
[25] Vinod K. Agarwal
"Multiple Fault Detection in PLAs".
IEEE transc. on comp. C_29, No.6, June, 1980.

[26] J. W. Gault, Robinson, S. M. Reddy
"Multiple Fault Detection in Combinational Networks".
IEEE transc. on comp. C_21, pp. 31-36, Jan. 1972.

[27] C. T. Ku, G. M. Masson
"The Boolean Difference and Multiple Fault Analysis".
IEEE transc. on comp. C_24, pp. 62-71, Jan. 1975.

[28] Janusz Rajski, Jerzy Tyszer
"Combinatorial Approach to Multiple Contact Faults in PLA".
IEEE transc. on comp. C_34, No.6, June. 1985.

[29] Janusz Rajski, Jerzy Tyszer
"The Influence of Masking Phenomenon on Coverage Capability
of Single Fault Test Sets in PLAs".
IEEE transc. on comp. C_35, No.1, Jan. 1986.

[30] K. S. Ramanatha
"A Design for Testability of Universal Crosspoint
Faults in PLAs".
IEEE Transc. on comp., C_32, No.6, June, 1983.

[31] Wilfried Daehn, Joachim Mucha
"A Hardware Approach to Self_Testing of Large PLAs".
IEEE Transc. on comp., C_30, No.11, Nov. 1981.

[32] Hideo Fujiwara, Kozo Kinoshita
"A Design of PLAs with Universal Tests".
IEEE Transc. on comp., C_30, No.11, Nov. 1981.

[33] C. Zheng, G. Musgrave
"A Functional Testable Design of PLAs".
IEE Electronic Design Automation conf. (EDA 1984).

[34] Javad Khakbaz
"A Testable PLA Design with Low Overhead and High
Fault Coverage".
IEEE Transc. on comp., C_33, No.8, Aug., 1984.

[35] Hideo Fujiwara
"A New PLA Design for Universal Testability".
IEEE Transc. on comp., C_33, No.8, Aug., 1984.

[36] Kewal K. Saluja, Kozo Kinoshita, Hideo Fujiwara
"An Easily Testable Design of PLA for Multiple Faults".
IEEE Transc. on comp., C_32, No.11, Nov., 1983.

[37] Saied Bozorgui_Nesbat, Edward J. McCluskey
"Lower Overhead Design for Testability of PLAs".
IEEE Transc. on comp., C_35, No.4, April, 1986.

[38] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen,
Alberto L. Sangiovanni_Vincentelli
""Logic Minimization Algorithms for VLSI Synthesis"
Kluwer Academic Pub. 1984.

APPENDIX A

A.1 The Cubic Notation

The most straightforward representation for a logic function is the 'tabular form' or 'truth table'. In this form, the function outputs are specified for each possible combination of the inputs. For example, the function

$F = \bar{X}\bar{Y}\bar{Z} + \bar{X}Y\bar{Z} + XY\bar{Z}$

is specified as follows

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

The above specification may also be mapped into a geometrical representation in which points in n_dimensional space are used to represent the possible binary codes or n_tuples. Karnaugh map may be considered as an attempt to project this n_dimensional space onto a 2_dimensional map (this is usually effective for up to 5 or 6 variables).

Accordingly, logic functions with 3 variables may be represented as a 3_dimensional unit cube as shown in Figure A.1. Each canonical product term (minterm) of the

function is associated with a unique point (vertex) of the cube. The cubical representation for the above function is illustrated in Figure A.2, where the indicated vertices represent the existence of the corresponding minterms in the original function. Planes and edges are used to represent the non_canonical product terms since each vertex is distance_one apart (i.e. only one variable changes its logical value between two adjacent vertices). For example, Figure A.3(a) represents the product term (Z), while Figure A.3(b) represents the product term (YZ).

Using the above notation, switching function expressions may be specified and manipulated as an arrays of n_tuples (cubes). Thus, a complete algebra may be established with defined operations on arrays of cubes to perform any computer manipulation on switching functions.

A.2 The Mathematical notion of sets

In this section, the notion of set and its basis operations, used in the material given in chapter 3, is introduced.

A set is simply a collection of objects without repetition. Each object in a set is called an element of that set. For example, a product term may be described as a set of integer numbers representing the decimal codes of its minterms. If an element, X, is a member of a set A, then it is written as $X \in A$ (read : X in A or X belongs to A), and if X is not an element in A then we write $X \notin A$ (read : X not in A or X does not belong to A).

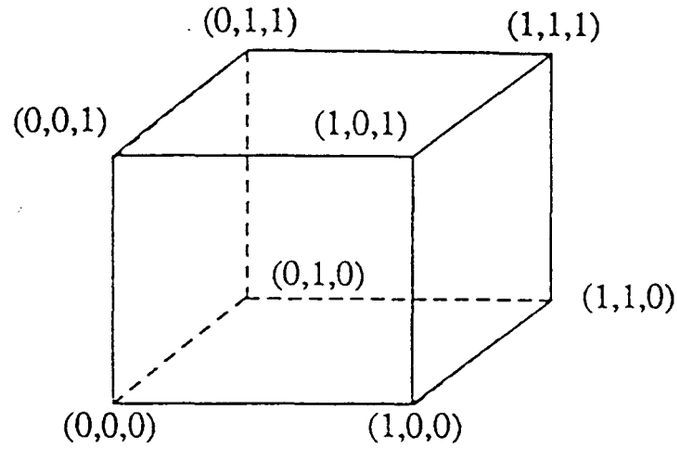


Fig. A.1

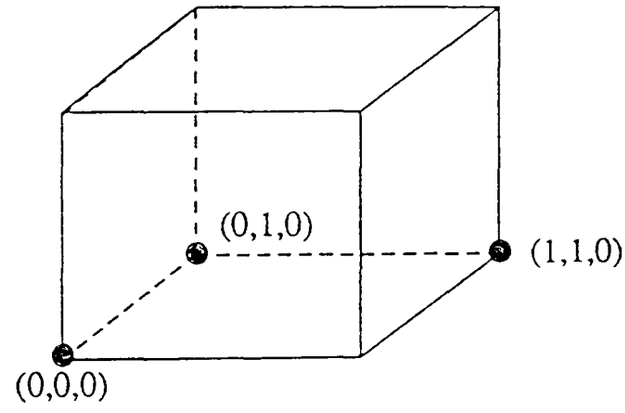


Fig. A.2

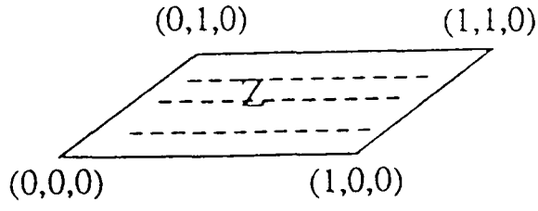


Fig. A.3(a)

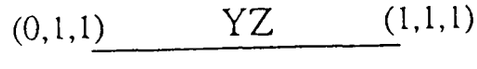


Fig. A.3(b)

Cubical Representation of Swithcing Functions

A set A is a subset of set B, written $A \subset B$, if and only if every element of A is a member of the set B. If A is not a subset of B, we write $A \not\subset B$. Then

$\{ 1,2,4 \} \subset \{ 1,2,3,4,5 \}$

and

$\{ 2,4,6 \} \not\subset \{ 1,2,3,4,5 \}$

The basic operations on sets are the binary operations, union ($\cup$), intersection ($\cap$), and difference (-).
If A and B are sets then these operations are defined as follows :

$A \cup B$: consists of all elements in either A or B,

$A \cap B$: consists of all elements in both A and B,

A - B : consists of all elements in A but not in B.

For example, if $A = \{ 0,1,3,5 \}$, $B = \{ 2,3,5 \}$ then

$A \cup B = \{ 0,1,2,3,5 \}$,

$A \cap B = \{ 3,5 \}$, and

$A - B = \{ 0,1 \}$.

APPENDIX B

Appendix B contains the detailed symbolic representation of some of the most important routines in FACTPLA program. Figures B1 (in the next two pages) illustrates the complete flow chart for deriving the decimal code (DecP) of a product term (BinP) in the manner required by the program. Figure B2 shows the derivation of the intersection vector (common minterms) between product terms P_1 and P_2. Finally, figure B3 illustrates the partitioning of a product term (P) of size (R) to obtain the Log_2R partitions. Each partition has two blocks, referred as block 0 and block 1 in the flow chart.

R : Size of the Product Term P

BinP : Product Term Cube

DecP : Decimal Code of BinP

binary & Temp : temprary Storage

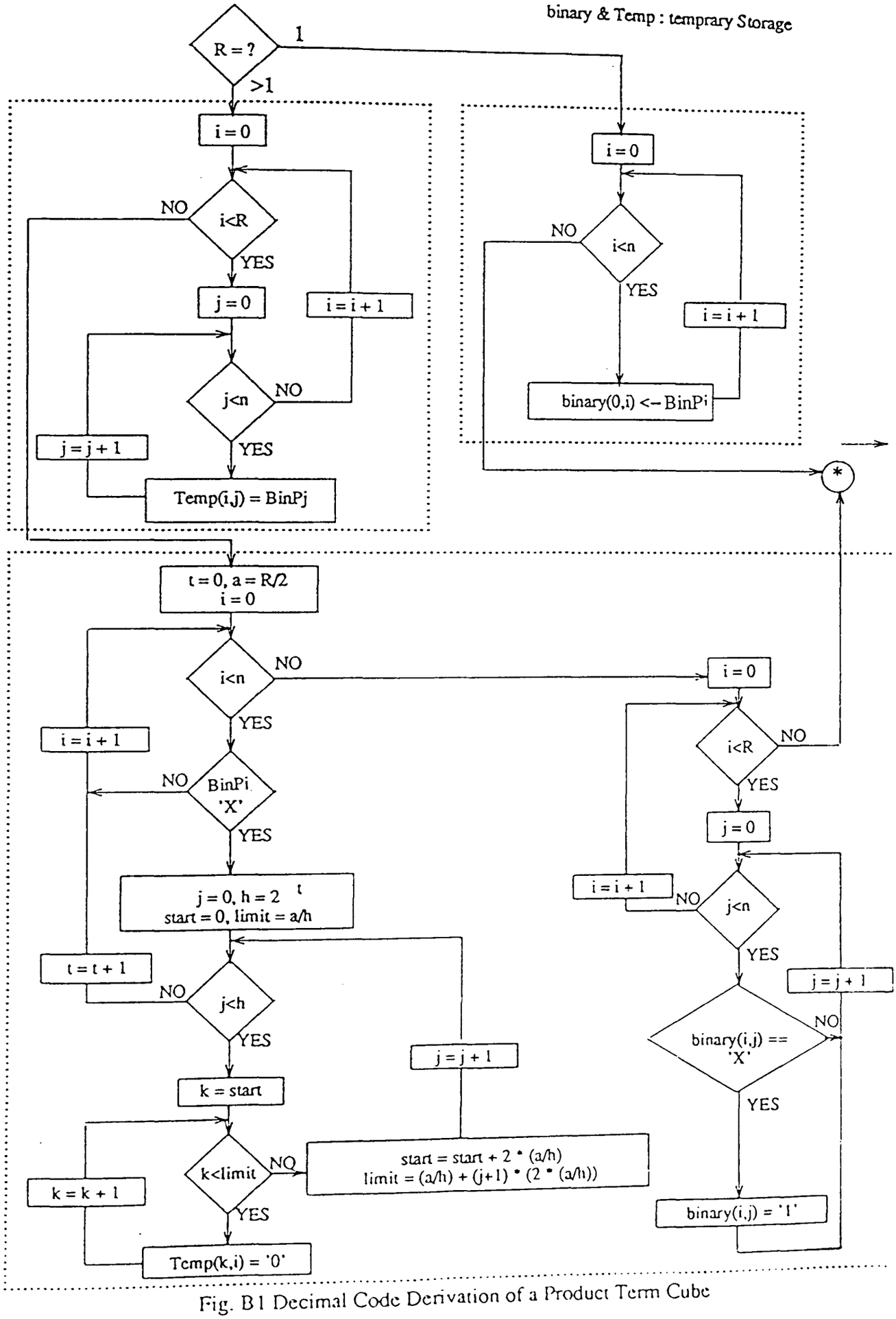


Fig. B1 Decimal Code Derivation of a Product Term Cube

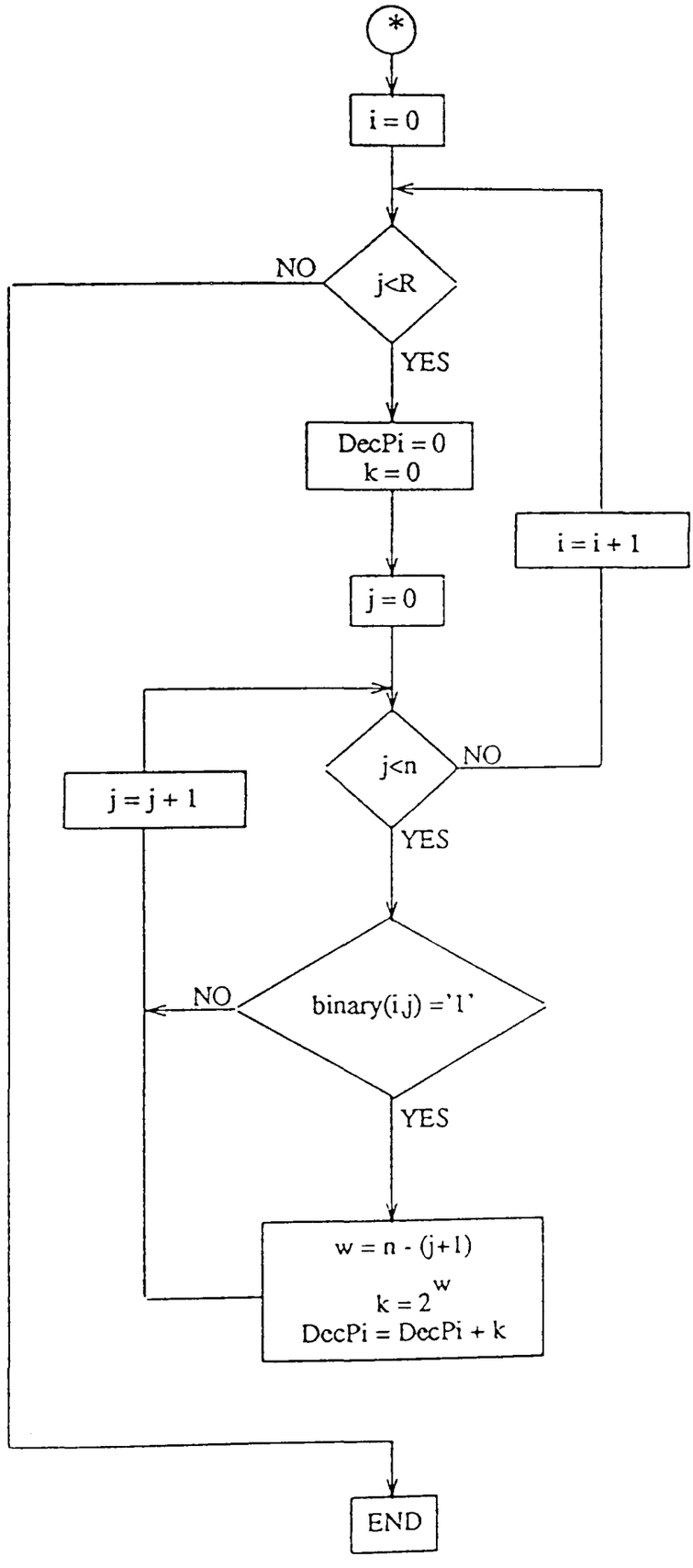


Fig. B1 Decimal Code Derivation of a Product Term Cube

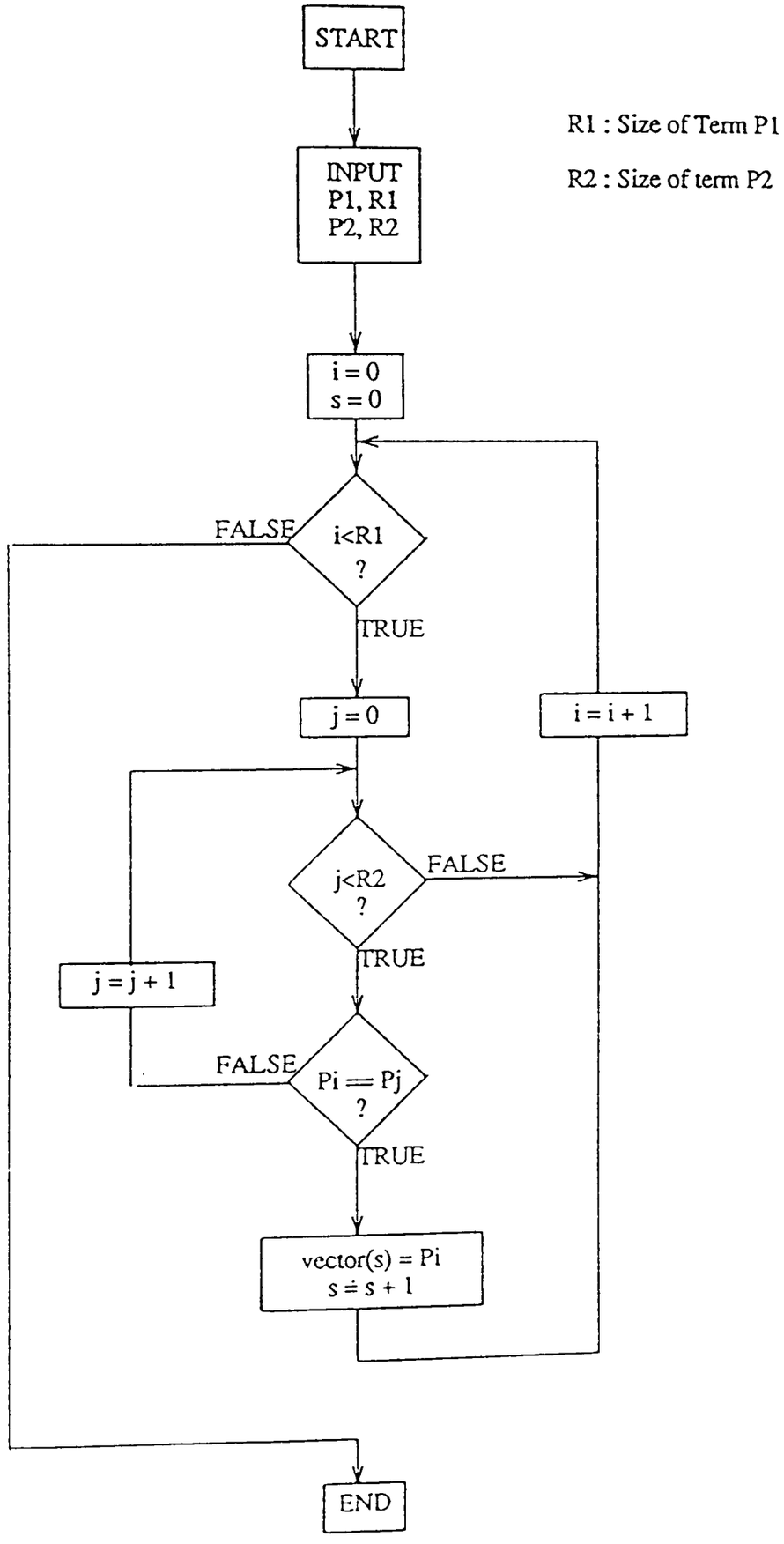


Fig. B2 Generation of the Intersection Vector between two terms

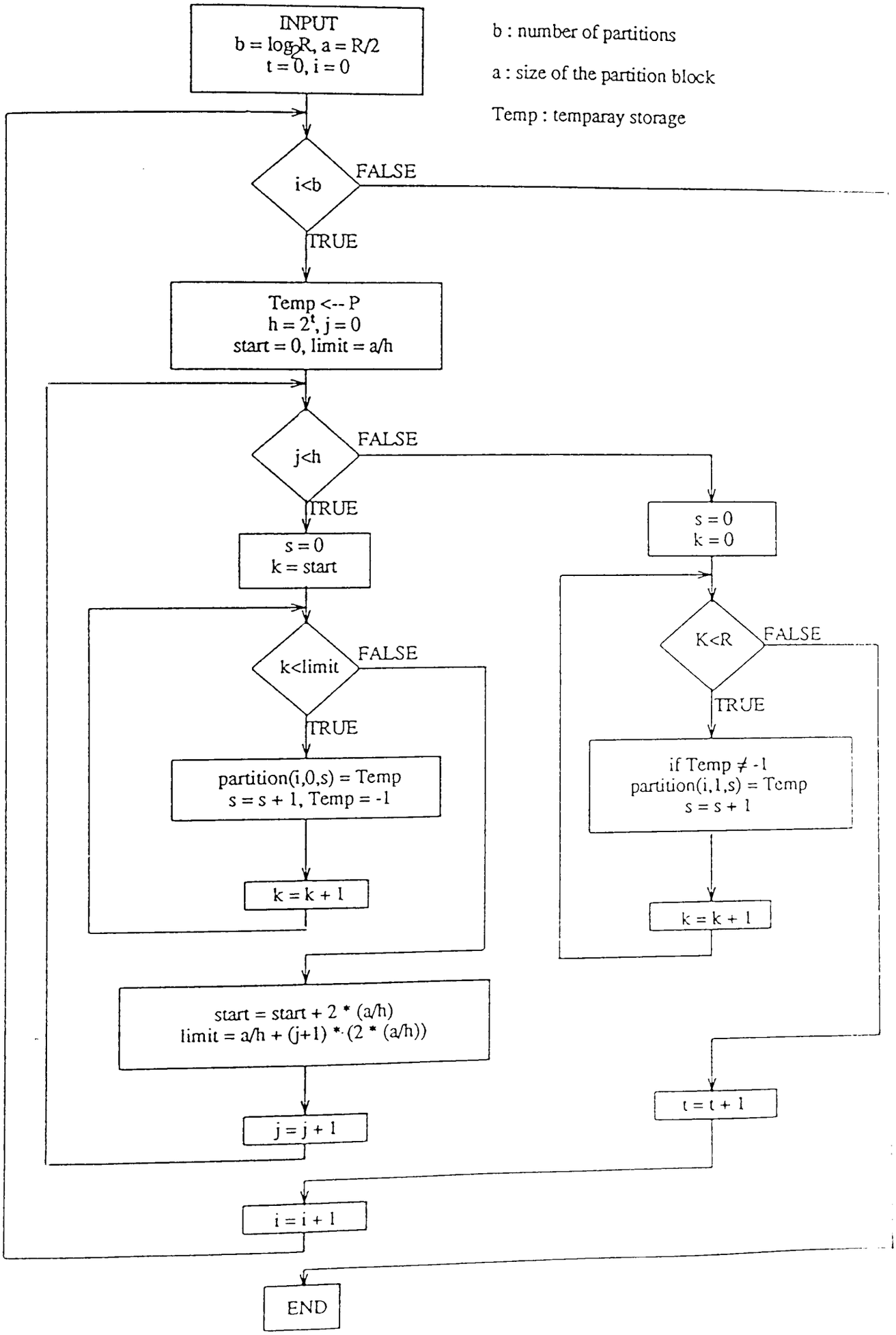


Fig. B3 Partitioning a Product Term (P) of Size (R)

APPENDIX C

Appendix C contains the results of PACTPLA on several PLA examples. The personality of each PLA is specified by 0, 1 , and X (don't care) in the input part of the product term (or row), and 1 and 0 (or~) in the output part. The locations of undetectable (growth, shrinkage, and appearance) faults are listed and the total values of the measure (MASK and RISK) are also given.

Note that the locations of faults are specified by m, n, f, and c where :

m_0 : used for the first row (product line),

n_0 : used for the first input variable (growth fault case),

f_0 : used for the first output column (function), and

c : used for input columns (shrinkage fault case) such that

c = 0 for the complemented bit line, and

c = 1 for the uncomplemented bit line.

```
.inputs 7
.outputs 10
.products 75
```

PLA_SX

```
X X X 0 X X X   ---------1-
X X X X 1 0 X   1---------
X X X X 0 1 0   -1--------
X X X X 1 0 1   -1--------
1 X X X 0 1 0   --1-------
0 1 X X X 0 1   ---1------
X 0 0 X X X 1   ----1-----
X 0 1 1 X X 0   ----1-----
X 1 0 0 X X 0   ----1-----
0 X 1 0 X X X   -----1----
1 X 1 1 X X X   -----1----
0 1 0 1 X X X   -----1----
1 X 0 0 X X X   -----1----
1 0 0 X X X X   -----1----
X 0 0 1 X X X   ------1---
X 1 X 0 X X X   ------1---
X 1 1 X X X X   ------1---
X X 0 1 X X X   -------1--
X X 1 0 X X X   -------1--
X X X X 1 1 1   ---------1
1 X X X 1 1 X   ---------1
0 0 X X 1 X 0   1---------
1 X X X 0 1 1   1---------
X 1 X X 0 1 1   1---------
1 1 X X X 0 1   -1--------
1 X 1 X X 0 1   -1--------
1 1 1 X 1 0 X   -1--------
1 1 X 1 1 0 X   -1--------
0 0 X X X 1 0   -1--------
0 X X X 0 0 1   --1-------
X 0 0 X 0 0 1   --1-------
1 1 1 X 0 X 0   --1-------
1 1 X 1 0 X 0   --1-------
0 X X X 1 0 0   --1-------
X 0 X X 1 0 0   --1-------
X X 0 0 1 0 0   --1-------
X 1 1 1 0 1 0   --1-------
0 0 X X X 1 0   ---1------
1 0 0 X X 0 X   ---1------
0 X 1 1 X 0 1   ---1------
0 1 1 1 X 0 X   ---1------
1 X 0 0 X 0 0   ---1------
1 1 1 X X 1 X   ---1------
1 1 X 1 X 1 X   ---1------
0 0 X 0 X X 1   ----1-----
X 1 1 1 X X 1   ----1-----
1 1 1 X X X 1   ----1-----
1 1 X 1 X X 1   ----1-----
1 0 1 X X X 0   ----1-----
X 1 1 1 1 1 X   ---------1
0 X X 0 1 X 0   1---------
0 X 0 X 1 X 0   1---------
X X 1 1 0 1 1   1---------
0 0 X 0 0 1 X   -1--------
0 0 0 X 0 1 X   -1--------
0 X X 0 X 1 0   -1--------
0 X 0 X X 1 0   -1--------
0 0 X 0 0 X 1   --1-------
0 0 0 X 0 X 1   --1-------
1 X X X 1 1 1   --1-------
X 1 X X 1 1 1   --1-------
0 0 1 1 1 1 X   --1-------
0 X X 0 1 X 0   --1-------
0 X 0 X 1 X 0   --1-------
1 1 X X 1 X 1   --1-------
0 X 0 X X 1 0   ---1------
1 0 X X X 0 0   ---1------
0 0 X 0 X 1 X   ---1------
0 0 0 X X 1 X   ---1------
1 1 X X X 1 1   ---1------
1 X 1 X X 1 1   ---1------
0 1 X 0 X X 0   ----1-----
0 1 0 X X X 0   ----1-----
```

App_fault location
(row , output)

```
( 60 , 9 )
( 59 , 9 )
( 73 , 8 )
( 69 , 8 )
( 66 , 8 )
( 62 , 8 )
( 57 , 8 )
( 55 , 8 )
( 53 , 8 )
( 50 , 8 )
( 44 , 8 )
( 41 , 8 )
( 35 , 8 )
( 18 , 8 )
( 15 , 8 )
( 12 , 8 )
( 9 , 8 )
( 8 , 8 )
( 14 , 7 )
( 11 , 7 )
( 9 , 7 )
( 73 , 6 )
( 49 , 6 )
( 46 , 6 )
( 45 , 6 )
( 42 , 6 )
( 40 , 6 )
( 36 , 6 )
( 31 , 6 )
( 26 , 6 )
( 8 , 6 )
( 41 , 5 )
( 38 , 5 )
( 58 , 4 )
( 57 , 4 )
( 30 , 4 )
( 56 , 3 )
( 55 , 3 )
( 54 , 3 )
( 53 , 3 )
( 28 , 3 )
( 51 , 2 )
( 50 , 2 )
( 21 , 2 )
( 67 , 1 )
( 66 , 1 )
( 37 , 1 )
( 36 , 1 )
( 4 , 1 )
( 63 , 0 )
( 62 , 0 )
( 35 , 0 )
( 34 , 0 )
( 33 , 0 )
( 27 , 0 )
( 26 , 0 )
( 3 , 0 )
```

Sh_fault location
(row , input , bit line)

```
( 49 , 6 , 0 )
( 49 , 0 , 0 )
( 20 , 6 , 0 )
( 19 , 0 , 0 )
( 16 , 3 , 1 )
( 15 , 2 , 0 )
( 13 , 3 , 1 )
( 12 , 1 , 1 )
( 74 , 3 , 1 )
( 73 , 2 , 1 )
( 48 , 3 , 0 )
( 47 , 2 , 0 )
( 46 , 3 , 0 )
( 45 , 0 , 0 )
( 44 , 2 , 1 )
( 8 , 0 , 1 )
( 7 , 0 , 0 )
( 72 , 1 , 0 )
( 71 , 3 , 0 )
( 71 , 2 , 0 )
( 70 , 6 , 1 )
( 70 , 3 , 1 )
( 69 , 6 , 1 )
( 69 , 2 , 1 )
( 68 , 2 , 1 )
( 67 , 3 , 1 )
( 67 , 1 , 1 )
( 66 , 2 , 1 )
( 66 , 1 , 1 )
( 43 , 6 , 0 )
( 43 , 2 , 0 )
( 42 , 6 , 0 )
( 42 , 3 , 0 )
( 41 , 1 , 1 )
( 40 , 6 , 0 )
( 39 , 1 , 0 )
( 38 , 6 , 1 )
( 37 , 3 , 1 )
( 37 , 2 , 1 )
( 65 , 5 , 0 )
( 65 , 1 , 0 )
( 64 , 5 , 0 )
( 64 , 2 , 0 )
( 63 , 5 , 1 )
( 63 , 3 , 1 )
( 62 , 5 , 1 )
( 62 , 2 , 1 )
( 60 , 0 , 0 )
( 59 , 2 , 0 )
( 59 , 1 , 0 )
( 58 , 5 , 1 )
( 58 , 3 , 1 )
( 57 , 5 , 1 )
( 57 , 2 , 1 )
( 36 , 0 , 0 )
( 35 , 1 , 1 )
( 35 , 0 , 1 )
( 34 , 0 , 1 )
```

```
( 33 , 3 , 1 )
( 33 , 2 , 1 )
( 33 , 1 , 1 )
( 32 , 5 , 0 )
( 32 , 2 , 0 )
( 31 , 5 , 0 )
( 31 , 3 , 0 )
( 30 , 0 , 1 )
( 56 , 4 , 1 )
( 56 , 3 , 1 )
( 56 , 1 , 1 )
( 55 , 4 , 1 )
( 55 , 2 , 1 )
( 55 , 1 , 1 )
( 54 , 6 , 1 )
( 54 , 3 , 1 )
( 53 , 6 , 1 )
( 53 , 2 , 1 )
( 28 , 4 , 1 )
( 28 , 3 , 1 )
( 28 , 2 , 1 )
( 27 , 6 , 0 )
( 27 , 2 , 0 )
( 26 , 6 , 0 )
( 26 , 3 , 0 )
( 25 , 4 , 0 )
( 25 , 1 , 0 )
( 24 , 4 , 0 )
( 24 , 2 , 0 )
( 52 , 1 , 0 )
( 52 , 0 , 0 )
( 51 , 5 , 1 )
( 51 , 3 , 1 )
( 51 , 1 , 1 )
( 50 , 5 , 1 )
( 50 , 2 , 1 )
( 50 , 1 , 1 )
( 23 , 0 , 0 )
( 22 , 1 , 0 )
( 21 , 5 , 1 )
( 21 , 3 , 1 )
( 21 , 2 , 1 )
```

App_faults = 57, a_s_mask = 112, a_s_risk = 114

Sh_faults = 200, s_g_mask = 2, s_a_mask = 191 (Total mask = 193)

```
.inputs   5
.outputs  28
.products 87
```

PLA BW

```
0 0 0 0 0  ----------------------------1
X 0 0 X 0  ------------------1---------
0 X 1 0 0  ---1----------1----1---------
1 X 0 X 0  ---1--------------1---------
1 X 0 1 X  ---1--1-----------1---------
0 X 0 0 1  ---1--------------1---------
X X 0 1 0  ---1----1----------1---------
X X 0 X X  ---------1--1---1-----------
0 X X X 0  -----1---1--1----------1---
0 0 X 0 X  ------------1---------------
X 1 0 X X  -1---1-----------------1---
X 1 0 X 0  --------1-----------------
0 X 1 0 X  -----1-------------------
X X 0 1 X  -----1-------------------
1 X 0 X X  -----1-------------------1-
0 X X 0 X  ----------------1-------1---
X X 0 X 1  ----------------------1-1-
X X 0 0 X  ----------------------1---
0 0 0 1 X  ---------------1----------
0 0 1 X 0  ----------------------1----
1 0 0 1 0  ----------------------1----
0 1 0 1 0  ----------------------1----
0 1 X X 0  -1-------------1----------
0 0 1 1 0  ----1---------------------
1 0 0 1 X  ----1---------------------
0 1 X 0 X  -1-------1----------------
0 X 0 1 0  --1-------------1---------
0 0 0 0 1  --1-----------------------
1 X 0 0 0  --1-----------------------
0 1 1 0 0  --------------------1------
0 1 0 0 0  ----------1--1-------------
0 0 1 0 0  --------1----1-----------1-
1 X 0 X 1  ------1-1----1------1-------
1 1 0 1 X  -------------1--------------
X 1 0 1 1  ------1-------------------
0 1 1 X 0  1-----1-------------------
X 0 0 0 0  -----------1--------------
0 X 1 1 0  1-----1--------1----1-1-----
0 0 X 0 0  -----------1--------------
1 1 0 X X  ------1-------------------
1 X 0 1 1  ----------------1-------1--
X 1 0 0 1  1-------------------------1--
X 0 0 1 1  ----------1-------------1--
0 X 1 X 0  -------1----------------1--
1 0 0 0 0  ----------------1-------1--
1 1 0 X 1  ----------------1----------
1 0 0 0 X  ----------------1----------
0 1 0 0 X  ---------------1-1----------
0 0 0 X 1  ----1----------1-----1-----
0 0 1 0 X  ----------------1----------
X 0 0 0 1  --------1--------1----------
X 1 0 1 X  ------------------------1-
0 0 0 1 0  ----------1-------1---------
X X 0 1 1  --------------------1-------
1 1 0 1 0  1---------------------------
1 X 0 1 0  ----1--1------------------
0 X X 0 0  -------1------------------
0 0 X X 0  -------1------------------
0 0 0 X 0  ----------1----------1-----
0 X 0 0 0  --------------------1-1-----
1 X 0 0 1  1---------------------1-----
0 X 0 1 1  ---------------1------1-----
1 1 0 1 1  ---------------1--------1----
0 X 1 0 1  1------------------1-1-----
1 1 0 0 0  ---------------1------------
0 0 0 1 1  ,---,---,---,---,---,---,---
1 1 0 0 1  --11---1--11------11--------
0 0 1 1 0  ---1------1---1----1-1----1-
0 1 1 1 0  --11------11--1---11------1-
0 0 1 0 1  --111-111-1--111-1-1-1-1-11-
0 1 1 0 1  --11-111-11-111-111-1-1-11-
0 0 0 0 0  1-1-1-1-1----111-1---1-1-1---
1 0 0 1 0  --1----------1-----1--------
1 1 0 1 0  --1--------1--1----1--------
0 1 1 0 0  ----------1------1-------1-
1 1 0 0 0  1---------1------1-1-------
1 0 0 1 1  1-1-----1---------------------
1 1 0 1 1  1-1--------1------1---------
0 0 1 0 0  --1-------------------1-------
1 0 0 0 0  1-------1-----------1-------
0 0 0 0 1  --------1------1-----1-------
0 1 0 1 0  1----------1--1---1-1-------
0 0 0 1 0  1----------1-----1-------
0 1 0 0 0  1----------1--1---1---1-------
0 1 0 0 1  ---------1---1---1---1---1-------
```

SEE FAULT DATA ->

```
( 73 , 27 )
( 86   26 )
( 83 , 26 )
( 82   26 )
( 81   26 )
( 80   26 )
( 79 , 26 )
( 78 , 26 )
( 77 , 26 )
( 75 , 26 )
( 74 , 26 )
( 68   26 )
( 67   26 )
( 66   26 )
( 65 , 26 )
( 64 , 26 )
( 63   26 )
( 62 , 26 )
( 61 , 26 )
( 60 , 26 )
( 55   26 )
( 54   26 )
( 53 , 26 )
( 50 , 26 )
( 49   26 )
( 48   26 )
( 46   26 )
( 45 , 26 )
( 44   26 )
( 43 , 26 )
( 42   26 )
( 41   26 )
( 40   26 )
( 39 , 26 )
( 37 , 26 )
( 35 , 26 )
( 34 , 26 )
( 33   26 )
( 32   26 )
( 29 , 26 )
( 28   26 )
( 27 , 26 )
( 24   26 )
( 23   26 )
( 21   26 )
( 20 , 26 )
( 19 , 26 )
( 12   26 )
( 5    26 )
( 4    26 )
( 3    26 )
( 2 , 26 )
( 86 , 25 )
( 81   25 )
( 80   25 )
( 79   25 )
( 78 , 25 )
( 76 , 25 )
( 70 , 25 )
( 69   25 )
( 68 , 25 )
( 65 , 25 )
( 63 , 25 )
( 62 , 25 )
( 49 , 25 )
( 45 , 25 )
( 38 , 25 )
( 37 , 25 )
( 36   25 )
( 35 , 25 )
( 31 , 25 )
( 29   25 )
( 23 , 25 )
( 19   25 )
( 12 , 25 )
( 2    25 )
( 0    25 )
( 86   24 )
( 85 , 24 )
( 84 , 24 )
( 83 , 24 )
( 82 , 24 )
( 81 , 24 )
( 80   24 )
( 79 , 24 )
( 78   24 )
( 77 , 24 )
( 76 , 24 )
( 75 , 24 )
( 73 , 24 )
( 72   24 )
( 71 , 24 )
( 70 , 24 )
( 69 , 24 )
( 68 , 24 )
( 67 , 24 )
( 66   24 )
( 65 , 24 )
( 64 , 24 )
( 63 , 24 )
( 62   24 )
( 61 , 24 )
( 60   24 )
( 59 , 24 )
( 58 , 24 )
( 57 , 24 )
( 56 , 24 )
( 54 , 24 )
( 53   24 )
( 52 , 24 )
( 51   24 )
( 50 , 24 )
( 49 , 24 )
( 48 , 24 )
( 47   24 )
( 46 , 24 )
( 45 , 24 )
( 44 , 24 )
( 43 , 24 )
( 42 , 24 )
( 41 , 24 )
( 40 , 24 )
( 39 , 24 )
( 38 , 24 )
( 37 , 24 )
( 36 , 24 )
( 35 , 24 )
( 34 , 24 )
( 33 , 24 )
( 32 , 24 )
( 31   24 )
( 30 , 24 )
( 29 , 24 )
( 28 , 24 )
( 27 , 24 )
( 26   24 )
( 25 , 24 )
( 23 , 24 )
( 22 , 24 )
( 21   24 )
( 19 , 24 )
( 18 , 24 )
( 12   24 )
( 11 , 24 )
( 9 , 24 )
( 5 , 24 )
( 2 , 24 )
( 0    24 )
( 83   23 )
( 82 , 23 )
( 80 , 23 )
( 79 , 23 )
( 74 , 23 )
( 69 , 23 )
( 65 , 23 )
( 63 , 23 )
( 49 , 23 )
( 38 , 23 )
( 31   23 )
( 27   23 )
( 23 , 23 )
( 9 , 23 )
( 0 , 23 )
( 85 , 22 )
( 84 , 22 )
( 73 , 22 )
( 72 , 22 )
( 71   22 )
( 70   22 )
( 69 , 22 )
( 68 , 22 )
( 67 , 22 )
( 66 , 22 )
( 65 , 22 )
( 52 , 22 )
( 30 , 22 )
( 23 , 22 )
( 18 , 22 )
( 0 , 22 )
( 76 , 21 )
( 63 , 21 )
( 61 , 21 )
( 59 , 21 )
( 47 , 21 )
( 30   21 )
( 25 , 21 )
( 23 , 21 )
( 0 , 21 )
( 85 , 20 )
( 79 , 20 )
( 78 , 20 )
( 73 , 20 )
( 72 , 20 )
( 71 , 20 )
( 70 , 20 )
( 69 , 20 )
( 68 , 20 )
( 67 , 20 )
( 66 , 20 )
( 65 , 20 )
( 64 , 20 )
( 62 , 20 )
( 61 , 20 )
( 60 , 20 )
( 58 , 20 )
( 57 , 20 )
( 52 , 20 )
( 50   20 )
( 49 , 20 )
( 48 , 20 )
( 46 , 20 )
( 45 , 20 )
( 44 , 20 )
( 42   20 )
( 40   20 )
( 38 , 20 )
( 36 , 20 )
( 34   20 )
( 31 , 20 )
( 30 , 20 )
( 28 , 20 )
( 27   20 )
( 26 , 20 )
( 24 , 20 )
( 23 , 20 )
( 21   20 )
( 20 , 20 )
( 19 , 20 )
( 18 , 20 )
( 9 , 20 )
( 1 , 20 )
( 0 , 20 )
( 86   19 )
( 84 , 19 )
( 83 , 19 )
( 82 , 19 )
( 31   19 )
```

CONTINUE FAULT DAT

(80 , 19)
(79 , 19)
(78 , 19)
(77 , 19)
(76 , 19)
(75 , 19)
(74 , 19)
(73 , 19)
(64 , 19)
(63 , 19)
(62 , 19)
(61 , 19)
(60 , 19)
(58 , 19)
(57 , 19)
(55 , 19)
(54 , 19)
(53 , 19)
(52 , 19)
(51 , 19)
(50 , 19)
(49 , 19)
(48 , 19)
(46 , 19)
(45 , 19)
(44 , 19)
(43 , 19)
(42 , 19)
(41 , 19)
(40 , 19)
(39 , 19)
(38 , 19)
(37 , 19)
(36 , 19)
(35 , 19)
(34 , 19)
(33 , 19)
(32 , 19)
(31 , 19)
(29 , 19)
(28 , 19)
(27 , 19)
(26 , 19)
(24 , 19)
(23 , 19)
(21 , 19)
(20 , 19)
(19 , 19)
(18 , 19)
(16 , 19)
(14 , 19)
(13 , 19)
(12 , 19)
(9 , 19)
(0 , 19)
(84 , 18)
(82 , 18)
(81 , 18)
(80 , 18)
(71 , 18)
(67 , 18)
(64 , 18)
(67 , 18)
(64 , 18)
(63 , 18)
(62 , 18)
(60 , 18)
(54 , 18)
(51 , 18)
(47 , 18)
(45 , 18)
(44 , 18)
(41 , 18)
(39 , 18)
(35 , 18)
(34 , 18)
(33 , 18)
(31 , 18)
(30 , 18)
(29 , 18)
(28 , 18)
(27 , 18)
(26 , 18)
(25 , 18)
(22 , 18)
(21 , 18)
(12 , 18)
(11 , 18)
(10 , 18)
(5 , 18)
(2 , 18)
(86 , 17)
(85 , 17)
(84 , 17)
(83 , 17)
(82 , 17)
(81 , 17)
(79 , 17)
(78 , 17)
(68 , 17)
(65 , 17)
(63 , 17)
(62 , 17)
(59 , 17)
(58 , 17)
(52 , 17)
(42 , 17)
(41 , 17)
(36 , 17)
(30 , 17)
(27 , 17)
(21 , 17)
(18 , 17)
(5 , 17)
(0 , 17)
(86 , 16)
(85 , 16)
(84 , 16)
(83 , 16)
(82 , 16)
(81 , 16)
(80 , 16)
(79 , 16)
(78 , 16)
(77 , 16)
(76 , 16)
(75 , 16)
(74 , 16)
(73 , 16)
(72 , 16)
(71 , 16)
(70 , 16)
(68 , 16)
(67 , 16)
(66 , 16)
(65 , 16)
(64 , 16)
(63 , 16)
(62 , 16)
(61 , 16)
(60 , 16)
(59 , 16)
(58 , 16)
(56 , 16)
(55 , 16)
(54 , 16)
(53 , 16)
(52 , 16)
(51 , 16)
(50 , 16)
(49 , 16)
(48 , 16)
(47 , 16)
(46 , 16)
(45 , 16)
(44 , 16)
(42 , 16)
(41 , 16)
(40 , 16)
(39 , 16)
(38 , 16)
(36 , 16)
(35 , 16)
(34 , 16)
(33 , 16)
(32 , 16)
(31 , 16)
(30 , 16)
(29 , 16)
(28 , 16)
(27 , 16)
(26 , 16)
(25 , 16)
(24 , 16)
(21 , 16)
(20 , 16)
(18 , 16)
(17 , 16)
(16 , 16)
(14 , 16)
(13 , 16)
(12 , 16)
(11 , 16)
(10 , 16)
(9 , 16)
(6 , 16)
(5 , 16)
(4 , 16)
(3 , 16)
(2 , 16)
(1 , 16)
(0 , 16)
(86 , 15)
(85 , 15)
(84 , 15)
(70 , 15)
(69 , 15)
(66 , 15)
(65 , 15)
(63 , 15)
(59 , 15)
(58 , 15)
(52 , 15)
(30 , 15)
(23 , 15)
(0 , 15)
(80 , 14)
(79 , 14)
(77 , 14)
(76 , 14)
(63 , 14)
(61 , 14)
(59 , 14)
(58 , 14)
(57 , 14)
(56 , 14)
(55 , 14)
(54 , 14)
(52 , 14)
(51 , 14)
(50 , 14)
(49 , 14)
(48 , 14)
(47 , 14)
(43 , 14)
(38 , 14)
(37 , 14)
(35 , 14)
(34 , 14)
(33 , 14)
(31 , 14)
(30 , 14)
(29 , 14)
(27 , 14)
(26 , 14)
(25 , 14)
(23 , 14)
(22 , 14)
(21 , 14)
(20 , 14)
(19 , 14)
(18 , 14)

CONTINUE FAULT DATA --->

(15 , 14)
(12 , 14)
(11 , 14)
(9 , 14)
(8 , 14)
(6 , 14)
(5 , 14)
(0 , 14)
(85 , 13)
(80 , 13)
(79 , 13)
(78 , 13)
(75 , 13)
(68 , 13)
(67 , 13)
(63 , 13)
(62 , 13)
(60 , 13)
(59 , 13)
(54 , 13)
(49 , 13)
(45 , 13)
(40 , 13)
(38 , 13)
(0 , 13)
(86 , 12)
(85 , 12)
(84 , 12)
(83 , 12)
(82 , 12)
(81 , 12)
(80 , 12)
(79 , 12)
(78 , 12)
(77 , 12)
(76 , 12)
(75 , 12)
(74 , 12)
(73 , 12)
(71 , 12)
(70 , 12)
(69 , 12)
(68 , 12)
(67 , 12)
(66 , 12)
(65 , 12)
(64 , 12)
(62 , 12)
(61 , 12)
(60 , 12)
(59 , 12)
(58 , 12)
(57 , 12)
(56 , 12)
(55 , 12)
(54 , 12)
(53 , 12)
(52 , 12)
(51 , 12)
(50 , 12)
(49 , 12)
(48 , 12)
(47 , 12)
(46 , 12)
(45 , 12)
(44 , 12)
(43 , 12)
(42 , 12)
(41 , 12)
(40 , 12)
(39 , 12)
(38 , 12)
(37 , 12)
(36 , 12)
(35 , 12)
(34 , 12)
(33 , 12)
(32 , 12)
(31 , 12)
(30 , 12)
(29 , 12)
(28 , 12)
(27 , 12)
(26 , 12)
(24 , 12)
(23 , 12)
(22 , 12)
(21 , 12)
(20 , 12)
(19 , 12)
(18 , 12)
(17 , 12)
(16 , 12)
(14 , 12)
(13 , 12)
(11 , 12)
(10 , 12)
(6 , 12)
(5 , 12)
(4 , 12)
(3 , 12)
(2 , 12)
(1 , 12)
(0 , 12)
(84 , 11)
(81 , 11)
(80 , 11)
(78 , 11)
(73 , 11)
(65 , 11)
(64 , 11)
(62 , 11)
(61 , 11)
(59 , 11)
(56 , 11)
(54 , 11)
(53 , 11)
(52 , 11)
(51 , 11)
(47 , 11)
(45 , 11)
(44 , 11)
(41 , 11)
(40 , 11)
(39 , 11)
(35 , 11)
(34 , 11)
(33 , 11)
(31 , 11)
(30 , 11)
(29 , 11)
(28 , 11)
(26 , 11)
(25 , 11)
(22 , 11)
(21 , 11)
(18 , 11)
(11 , 11)
(10 , 11)
(2 , 11)
(0 , 11)
(85 , 10)
(84 , 10)
(63 , 10)
(61 , 10)
(60 , 10)
(37 , 10)
(23 , 10)
(18 , 10)
(86 , 9)
(85 , 9)
(84 , 9)
(83 , 9)
(82 , 9)
(81 , 9)
(80 , 9)
(79 , 9)
(78 , 9)
(77 , 9)
(76 , 9)
(75 , 9)
(74 , 9)
(73 , 9)
(72 , 9)
(70 , 9)
(69 , 9)
(68 , 9)
(67 , 9)
(66 , 9)
(65 , 9)
(64 , 9)
(62 , 9)
(61 , 9)
(60 , 9)
(59 , 9)
(58 , 9)
(57 , 9)
(56 , 9)
(55 , 9)
(54 , 9)
(53 , 9)
(52 , 9)
(51 , 9)
(50 , 9)
(48 , 9)
(47 , 9)
(46 , 9)
(45 , 9)
(44 , 9)
(43 , 9)
(42 , 9)
(41 , 9)
(40 , 9)
(39 , 9)
(38 , 9)
(37 , 9)
(36 , 9)
(35 , 9)
(34 , 9)
(33 , 9)
(32 , 9)
(31 , 9)
(30 , 9)
(29 , 9)
(28 , 9)
(27 , 9)
(26 , 9)
(24 , 9)
(23 , 9)
(22 , 9)
(21 , 9)
(20 , 9)
(19 , 9)
(18 , 9)
(17 , 9)
(16 , 9)
(14 , 9)
(13 , 9)
(11 , 9)
(10 , 9)
(6 , 9)
(5 , 9)
(4 , 9)
(3 , 9)
(2 , 9)
(1 , 9)
(0 , 9)
(85 , 8)
(84 , 8)
(83 , 8)
(82 , 8)
(80 , 8)
(79 , 8)
(78 , 8)
(77 , 8)
(75 , 8)
(74 , 8)
(68 , 8)
(67 , 8)
(64 , 8)

CONTINUE FAULT DATA -->>

(63 , 8)
(62 , 8)
(60 , 8)
(59 , 8)
(58 , 8)
(55 , 8)
(54 , 8)
(52 , 8)
(49 , 8)
(45 , 8)
(40 , 8)
(39 , 8)
(38 , 8)
(33 , 8)
(30 , 8)
(27 , 8)
(26 , 8)
(24 , 8)
(21 , 8)
(20 , 8)
(9 , 8)
(4 , 8)
(0 , 8)
(85 , 7)
(84 , 7)
(80 , 7)
(76 , 7)
(75 , 7)
(74 , 7)
(73 , 7)
(70 , 7)
(69 , 7)
(63 , 7)
(61 , 7)
(60 , 7)
(59 , 7)
(58 , 7)
(54 , 7)
(52 , 7)
(50 , 7)
(49 , 7)
(48 , 7)
(47 , 7)
(46 , 7)
(44 , 7)
(42 , 7)
(41 , 7)
(38 , 7)
(37 , 7)
(36 , 7)
(35 , 7)
(31 , 7)
(30 , 7)
(29 , 7)
(27 , 7)
(25 , 7)
(24 , 7)
(23 , 7)
(20 , 7)
(19 , 7)

(18 , 7)
(15 , 7)
(12 , 7)
(9 , 7)
(5 , 7)
(2 , 7)
(1 , 7)
(0 , 7)
(79 , 6)
(78 , 6)
(77 , 6)
(76 , 6)
(75 , 6)
(74 , 6)
(70 , 6)
(69 , 6)
(68 , 6)
(67 , 6)
(66 , 6)
(64 , 6)
(63 , 6)
(62 , 6)
(60 , 6)
(55 , 6)
(54 , 6)
(45 , 6)
(40 , 6)
(33 , 6)
(29 , 6)
(24 , 6)
(23 , 6)
(20 , 6)
(0 , 6)
(86 , 5)
(85 , 5)
(84 , 5)
(83 , 5)
(81 , 5)
(80 , 5)
(79 , 5)
(78 , 5)
(77 , 5)
(76 , 5)
(75 , 5)
(74 , 5)
(73 , 5)
(72 , 5)
(71 , 5)
(70 , 5)
(69 , 5)
(68 , 5)
(67 , 5)
(66 , 5)
(65 , 5)
(64 , 5)
(63 , 5)
(62 , 5)
(61 , 5)
(60 , 5)
(59 , 5)

(58 , 5)
(57 , 5)
(56 , 5)
(55 , 5)
(54 , 5)
(53 , 5)
(52 , 5)
(51 , 5)
(49 , 5)
(47 , 5)
(46 , 5)
(45 , 5)
(44 , 5)
(43 , 5)
(42 , 5)
(41 , 5)
(40 , 5)
(39 , 5)
(38 , 5)
(37 , 5)
(36 , 5)
(35 , 5)
(34 , 5)
(33 , 5)
(32 , 5)
(31 , 5)
(30 , 5)
(29 , 5)
(28 , 5)
(26 , 5)
(25 , 5)
(24 , 5)
(23 , 5)
(22 , 5)
(21 , 5)
(20 , 5)
(19 , 5)
(18 , 5)
(11 , 5)
(6 , 5)
(4 , 5)
(3 , 5)
(2 , 5)
(1 , 5)
(0 , 5)
(82 , 4)
(78 , 4)
(75 , 4)
(74 , 4)
(69 , 4)
(65 , 4)
(63 , 4)
(54 , 4)
(42 , 4)
(27 , 4)
(20 , 4)
(0 , 4)
(86 , 3)
(84 , 3)
(83 , 3)

(82 , 3)
(81 , 3)
(80 , 3)
(79 , 3)
(78 , 3)
(77 , 3)
(76 , 3)
(75 , 3)
(74 , 3)
(64 , 3)
(63 , 3)
(62 , 3)
(61 , 3)
(60 , 3)
(55 , 3)
(54 , 3)
(53 , 3)
(52 , 3)
(51 , 3)
(50 , 3)
(49 , 3)
(48 , 3)
(46 , 3)
(45 , 3)
(44 , 3)
(43 , 3)
(42 , 3)
(41 , 3)
(40 , 3)
(39 , 3)
(37 , 3)
(35 , 3)
(34 , 3)
(33 , 3)
(32 , 3)
(31 , 3)
(29 , 3)
(28 , 3)
(27 , 3)
(26 , 3)
(24 , 3)
(23 , 3)
(21 , 3)
(20 , 3)
(19 , 3)
(18 , 3)
(16 , 3)
(14 , 3)
(13 , 3)
(12 , 3)
(84 , 2)
(83 , 2)
(82 , 2)
(81 , 2)
(77 , 2)
(64 , 2)
(63 , 2)
(62 , 2)
(60 , 2)
(58 , 2)

(55 , 2)
(54 , 2)
(52 , 2)
(50 , 2)
(49 , 2)
(46 , 2)
(45 , 2)
(44 , 2)
(40 , 2)
(39 , 2)
(38 , 2)
(36 , 2)
(33 , 2)
(32 , 2)
(31 , 2)
(24 , 2)
(21 , 2)
(20 , 2)
(14 , 2)
(9 , 2)
(6 , 2)
(4 , 2)
(3 , 2)
(1 , 2)
(0 , 2)
(86 , 1)
(85 , 1)
(83 , 1)
(79 , 1)
(77 , 1)
(76 , 1)
(75 , 1)
(72 , 1)
(70 , 1)
(68 , 1)
(66 , 1)
(64 , 1)
(62 , 1)
(54 , 1)
(51 , 1)
(47 , 1)
(45 , 1)
(41 , 1)
(39 , 1)
(35 , 1)
(34 , 1)
(33 , 1)
(30 , 1)
(29 , 1)
(21 , 1)
(11 , 1)
(86 , 0)
(76 , 0)
(75 , 0)
(72 , 0)
(71 , 0)
(70 , 0)
(69 , 0)
(68 , 0)
(67 , 0)

CONTINUE FAULT DATA -->

(64 , 0)
(62 , 0)
(61 , 0)
(59 , 0)
(58 , 0)
(53 , 0)
(52 , 0)
(51 , 0)
(47 , 0)
(46 , 0)
(45 , 0)
(44 , 0)
(42 , 0)
(40 , 0)
(39 , 0)
(36 , 0)
(34 , 0)
(33 , 0)
(32 , 0)
(30 , 0)
(29 , 0)
(28 , 0)
(26 , 0)
(25 , 0)
(23 , 0)
(22 , 0)
(21 , 0)
(18 , 0)
(11 , 0)
(10 , 0)
(0 , 0)

App_faults = 1048

a_s_mask = 220

ag_s_risk = 22

Gr fault location
(row , input)

(54 , 4)
(54 , 3)
(54 , 0)
(65 , 1)
(77 , 4)
(77 , 1)
(77 , 0)
(78 , 3)
(79 , 4)
(79 , 3)
(81 , 4)
(81 , 0)
(83 , 4)
(83 , 3)
(83 , 2)
(83 , 1)
(84 , 4)
(84 , 3)
(84 , 2)
(84 , 1)
(85 , 4)
(85 , 2)
(27 , 4)
(27 , 2)
(27 , 0)
(28 , 4)
(28 , 3)
(71 , 1)
(74 , 0)
(75 , 4)
(75 , 3)
(75 , 0)
(80 , 4)
(80 , 2)
(2 , 4)
(2 , 3)
(3 , 4)
(5 , 3)
(5 , 2)
(5 , 0)
(56 , 4)
(82 , 4)
(82 , 3)
(82 , 2)
(82 , 0)
(86 , 4)
(86 , 3)
(86 , 2)
(31 , 4)
(36 , 1)
(38 , 1)
(76 , 4)
(76 , 3)
(76 , 1)
(64 , 3)
(64 , 0)
(44 , 0)
(46 , 1)
(49 , 1)
(1 , 4)
(29 , 4)
(29 , 2)

Gr_faults = 62 g_s_mask = 189

Sh fault location
(row , input , bit line)

(35 , 3 , 0)
(22 , 2 , 1)
(22 , 3 , 1)
(25 , 2 , 1)
(25 , 4 , 1)
(3 , 3 , 0)
(24 , 4 , 1)
(12 , 4 , 1)
(13 , 0 , 0)
(13 , 1 , 0)
(13 , 4 , 1)
(34 , 0 , 0)
(39 , 3 , 0)
(39 , 4 , 0)
(56 , 1 , 1)
(56 , 2 , 0)
(57 , 2 , 0)
(57 , 3 , 1)
(11 , 3 , 0)
(50 , 0 , 0)
(36 , 0 , 1)
(38 , 2 , 1)
(58 , 3 , 1)
(9 , 2 , 1)
(9 , 4 , 1)
(33 , 4 , 0)
(18 , 4 , 0)
(15 , 2 , 1)
(45 , 3 , 0)
(46 , 4 , 0)
(1 , 0 , 0)
(1 , 3 , 0)
(53 , 0 , 0)
(17 , 0 , 1)
(17 , 1 , 0)
(17 , 4 , 0)
(51 , 0 , 0)
(51 , 4 , 0)

~~Sh faults = 38~~ s_g_mask = 16 s_a_mask = 650 (Total mask = 666)
s_g_risk = 21

.inputs 5
.outputs 28 PLA BW1
.products 85

```
0 0 0 0 0  ----------------------------1
X 0 0 X 0  --------------------1--------
0 X 1 0 0  ---1----------1----1--------
1 X 0 X 0  ---1----------------1--------
1 X 0 1 X  ---1--1-------------1--------
0 X 0 0 1  ---1----------------1--------
X X 0 1 0  ---1----1----------1--------
X X 0 X X  ----------1--1---1----------
0 X X X 0  -----1---1--1-----------1---
0 0 X 0 X  ------------1--------------
X 1 0 X X  -1---1-------------------1---
X 1 0 X 0  --------1-----------------
0 X 1 0 X  -----1----------------------
X X 0 1 X  -----1----------------------
1 X 0 X X  -----1----------------------1
0 X X 0 X  ----------------1-------1---
X X 0 X 1  ------------------------1-1-
X X 0 0 X  ------------------------1---
0 0 0 1 X  ----------------1-----------
0 0 1 X 0  ------------------------1----
1 0 0 1 0  ------------------------1----
0 1 0 1 0  ------------------------1----
0 1 X X 0  -1--------------1-----------
0 0 1 1 0  ----1-----------------------
1 0 0 1 X  ----1-----------------------
0 1 X 0 X  -1-------1------------------
0 X 0 1 0  --1--------------1----------
0 0 0 0 1  --1-------------------------
1 X 0 0 0  --1-------------------------
0 1 1 0 0  --------------------1------
0 1 0 0 0  ----------1--1---------------
0 0 1 0 0  --------1----1-------------1-
1 X 0 X 1  ------1-1----1------1-------
1 1 0 1 X  ------------1----------------
X 1 0 1 1  ------1-----------------------
0 1 1 X 0  1-----1-----------------------
X 0 0 0 0  ----------1-------------------
0 X 1 1 0  1-----1---------1----1-1-----
0 0 X 0 0  ----------1-------------------
1 1 0 X X  ------1-----------------------
1 X 0 1 1  ----------------1-------1---
X 1 0 0 1  1-------------------------1---
X 0 0 1 1  ----------1-------------1---
0 X 1 X 0  -------1--------------------1---
1 0 0 0 0  ----------------1-------1---
1 1 0 X 1  ----------------1-----------
1 0 0 0 X  ------------------1---------
0 1 0 0 X  ----------------1-1---------
0 0 0 X 1  ----1-----------1-----1-----
0 0 1 0 X  ----------------1-----------
X 0 0 0 1  --------1---------1---------
X 1 0 1 X  ------------------------------1-
0 0 0 1 0  ----------1-------1----------
X X 0 1 1  ------------------1----------
1 1 0 1 0  1-----------------------------
1 X 0 1 0  ----1--1----------------------
0 X X 0 0  -------1----------------------
0 0 X X 0  -------1----------------------
0 0 0 X 0  ----------1---------1-----
0 X 0 0 0  ----------------------1-1-----
1 X 0 0 1  1---------------------1-----
0 X 0 1 1  ----------------1-------1----
1 1 0 1 1  ----------------------1-1-----
0 X 1 0 1  1-----------------------------
1 1 0 0 0  ----------------1-------------
```

SEE FAULT DATA -->>

App_fault location
(row , output)
==========

(64 , 26)
(62 , 26)
(61 , 26)
(60 , 26)
(55 , 26)
(54 , 26)
(53 , 26)
(50 , 26)
(48 , 26)
(46 , 26)
(45 , 26)
(44 , 26)
(42 , 26)
(41 , 26)
(40 , 26)
(39 , 26)
(34 , 26)
(33 , 26)
(32 , 26)
(28 , 26)
(27 , 26)
(24 , 26)
(21 , 26)
(20 , 26)
(5 , 26)
(4 , 26)
(3 , 26)
(62 , 25)
(45 , 25)
(37 , 25)
(35 , 25)
(31 , 25)
(29 , 25)
(23 , 25)
(19 , 25)
(2 , 25)
(64 , 24)
(63 , 24)
(62 , 24)
(61 , 24)
(60 , 24)
(59 , 24)
(58 , 24)
(57 , 24)
(56 , 24)
(54 , 24)
(53 , 24)
(52 , 24)
(51 , 24)
(50 , 24)
(49 , 24)
(48 , 24)
(47 , 24)
(46 , 24)
(45 , 24)
(44 , 24)
(43 , 24)
(42 , 24)
(41 , 24)
(40 , 24)
(39 , 24)
(38 , 24)
(37 , 24)
(36 , 24)
(35 , 24)
(34 , 24)
(33 , 24)
(32 , 24)
(31 , 24)
(30 , 24)
(29 , 24)
(28 , 24)
(27 , 24)
(26 , 24)
(25 , 24)
(23 , 24)
(22 , 24)
(21 , 24)
(19 , 24)
(18 , 24)
(12 , 24)
(11 , 24)
(9 , 24)
(5 , 24)
(2 , 24)
(0 , 24)
(31 , 23)
(27 , 23)
(23 , 23)
(52 , 22)
(30 , 22)
(23 , 22)
(18 , 22)
(0 , 22)
(62 , 20)
(61 , 20)
(60 , 20)
(45 , 20)
(42 , 20)
(40 , 20)
(34 , 20)
(30 , 20)
(23 , 20)
(0 , 20)
(64 , 19)
(62 , 19)
(58 , 19)
(55 , 19)
(54 , 19)
(52 , 19)
(44 , 19)
(40 , 19)
(38 , 19)
(36 , 19)
(33 , 19)
(31 , 19)
(29 , 19)
(28 , 19)
(27 , 19)
(26 , 19)
(24 , 19)
(21 , 19)
(20 , 19)
(0 , 19)
(44 , 18)
(31 , 18)
(27 , 18)
(62 , 17)
(52 , 17)
(42 , 17)
(41 , 17)
(30 , 17)
(27 , 17)
(21 , 17)
(18 , 17)
(5 , 17)
(64 , 16)
(63 , 16)
(62 , 16)
(61 , 16)
(60 , 16)
(59 , 16)
(58 , 16)
(56 , 16)
(55 , 16)
(54 , 16)
(53 , 16)
(52 , 16)
(51 , 16)
(50 , 16)
(49 , 16)
(48 , 16)
(47 , 16)
(46 , 16)
(45 , 16)
(44 , 16)
(42 , 16)
(41 , 16)
(40 , 16)
(39 , 16)
(38 , 16)
(36 , 16)
(35 , 16)
(34 , 16)
(33 , 16)
(32 , 16)
(31 , 16)
(30 , 16)
(29 , 16)
(28 , 16)
(27 , 16)
(26 , 16)
(25 , 16)
(24 , 16)
(21 , 16)
(20 , 16)
(18 , 16)
(17 , 16)
(16 , 16)
(14 , 16)
(13 , 16)
(12 , 16)
(11 , 16)
(10 , 16)
(9 , 16)
(6 , 16)
(5 , 16)
(4 , 16)
(3 , 16)
(2 , 16)
(1 , 16)
(0 , 16)
(52 , 15)
(30 , 15)
(23 , 15)
(31 , 14)
(29 , 14)
(62 , 13)
(60 , 13)
(54 , 13)
(45 , 13)
(40 , 13)
(64 , 12)
(62 , 12)
(61 , 12)
(60 , 12)
(59 , 12)
(58 , 12)
(57 , 12)
(56 , 12)
(55 , 12)
(54 , 12)
(53 , 12)
(52 , 12)
(51 , 12)
(50 , 12)
(49 , 12)
(48 , 12)
(47 , 12)
(46 , 12)
(45 , 12)
(44 , 12)
(43 , 12)
(42 , 12)
(41 , 12)
(40 , 12)
(39 , 12)
(38 , 12)
(37 , 12)
(36 , 12)
(35 , 12)
(34 , 12)
(33 , 12)
(32 , 12)
(31 , 12)
(30 , 12)
(29 , 12)
(28 , 12)
(27 , 12)
(26 , 12)
(24 , 12)
(23 , 12)
(22 , 12)
(21 , 12)
(20 , 12)
(19 , 12)
(18 , 12)
(17 , 12)
(16 , 12)
(14 , 12)
(13 , 12)
(11 , 12)
(10 , 12)
(6 , 12)
(5 , 12)
(4 , 12)
(3 , 12)
(2 , 12)
(1 , 12)
(0 , 12)
(52 , 11)
(44 , 11)
(31 , 11)
(18 , 11)
(0 , 11)
(64 , 9)
(62 , 9)
(61 , 9)
(60 , 9)
(59 , 9)
(58 , 9)
(57 , 9)
(56 , 9)
(55 , 9)
(54 , 9)
(53 , 9)
(52 , 9)
(51 , 9)
(50 , 9)
(48 , 9)
(47 , 9)
(46 , 9)
(45 , 9)
(44 , 9)
(43 , 9)
(42 , 9)
(41 , 9)
(40 , 9)
(39 , 9)
(38 , 9)
(37 , 9)
(36 , 9)
(35 , 9)
(34 , 9)
(33 , 9)
(32 , 9)
(31 , 9)

CONTINUE FAULT DATA -->

(30 , 9)
(29 , 9)
(28 , 9)
(27 , 9)
(26 , 9)
(24 , 9)
(23 , 9)
(22 , 9)
(21 , 9)
(20 , 9)
(19 , 9)
(18 , 9)
(17 , 9)
(16 , 9)
(14 , 9)
(13 , 9)
(11 , 9)
(10 , 9)
(6 , 9)
(5 , 9)
(4 , 9)
(3 , 9)
(2 , 9)
(1 , 9)
(0 , 9)
(64 , 8)
(62 , 8)
(60 , 8)
(55 , 8)
(54 , 8)
(52 , 8)
(45 , 8)
(40 , 8)
(39 , 8)
(33 , 8)
(30 , 8)
(27 , 8)
(26 , 8)
(24 , 8)
(21 , 8)
(20 , 8)
(4 , 8)
(59 , 7)
(58 , 7)
(54 , 7)
(52 , 7)
(38 , 7)
(37 , 7)
(35 , 7)
(31 , 7)
(30 , 7)
(29 , 7)
(23 , 7)
(20 , 7)
(19 , 7)
(2 , 7)
(0 , 7)
(64 , 6)
(62 , 6)
(60 , 6)
(55 , 6)
(54 , 6)
(45 , 6)
(40 , 6)
(33 , 6)
(29 , 6)
(24 , 6)
(23 , 6)
(20 , 6)
(64 , 5)
(63 , 5)
(62 , 5)
(61 , 5)
(60 , 5)
(59 , 5)
(58 , 5)
(57 , 5)
(56 , 5)
(55 , 5)
(54 , 5)
(53 , 5)
(52 , 5)
(51 , 5)
(49 , 5)
(47 , 5)
(46 , 5)
(45 , 5)
(44 , 5)
(43 , 5)
(42 , 5)
(41 , 5)
(40 , 5)
(39 , 5)
(38 , 5)
(37 , 5)
(36 , 5)
(35 , 5)
(34 , 5)
(33 , 5)
(32 , 5)
(31 , 5)
(30 , 5)
(29 , 5)
(28 , 5)
(26 , 5)
(25 , 5)
(24 , 5)
(23 , 5)
(22 , 5)
(21 , 5)
(20 , 5)
(19 , 5)
(18 , 5)
(11 , 5)
(6 , 5)
(4 , 5)
(3 , 5)
(2 , 5)
(1 , 5)
(0 , 5)
(54 , 4)
(42 , 4)
(27 , 4)
(20 , 4)
(64 , 3)
(62 , 3)
(55 , 3)
(54 , 3)
(52 , 3)
(44 , 3)
(40 , 3)
(33 , 3)
(31 , 3)
(29 , 3)
(28 , 3)
(27 , 3)
(26 , 3)
(24 , 3)
(21 , 3)
(20 , 3)
(64 , 2)
(52 , 2)
(44 , 2)
(21 , 2)
(64 , 1)
(62 , 1)
(54 , 1)
(51 , 1)
(47 , 1)
(45 , 1)
(41 , 1)
(39 , 1)
(35 , 1)
(34 , 1)
(33 , 1)
(30 , 1)
(29 , 1)
(21 , 1)
(11 , 1)
(29 , 0)
(23 , 0)

CONTINUE FAULT DATA -->

Sh_fault location
(row input bit line)
==========

(35 3 0)
(22 2 1)
(22 3 1)
(25 2 1)
(25 4 1)
(3 3 0)
(24 4 1)
(12 4 1)
(13 0 0)
(13 1 0)
(13 4 1)
(34 0 0)
(39 3 0)
(39 4 0)
(56 1 1)
(56 2 0)
(57 2 0)
(57 3 1)
(11 3 0)
(50 0 0)
(36 0 1)
(38 2 1)
(58 3 1)
(9 2 1)
(9 4 1)
(33 4 0)
(18 4 0)
(15 2 1)
(45 3 0)
(46 4 0)
(1 0 0)
(1 3 0)
(53 0 0)
(17 0 1)
(17 1 0)
(17 4 0)
(51 0 0)
(51 4 0)

Sh_faults = 38 s_g_mask = 8 s_a_mask = 354 (Total mask = 362)
s_g_risk = 29

App_faults = 458, a_s_mask = 149, a_s_risk = 22

```
.inputs  7
.outputs  2      PLA CON1
.products  9

X 1 X X 1 X X  1 0
1 X 1 1 X X X  1 0
X 0 0 1 X X X  1 0
0 1 X X X 1 X  1 0
X 0 X X 0 X X  0 1
1 X X X 0 X X  0 1
0 X X X X X 0  0 1
0 1 X X 1 X X  0 1
1 0 X 0 X X X  0 1
```

```
Sh_fault location
( row   input . bit line )
==========================
( 8 . 4 , 1 )
( 7 . 6 . 1 )
( 5 . 1 . 1 )
( 4 . 6 . 1 )
( 4   0 . 0 )
( 3   4 . 0 )

Sh_faults = 6, s_g_mask = 0  s_a_mask = 1 (Total mask = 1)
               s_g_risk = 22
```

```
App fault location
( row , output )
==============
( 7   0 )

App_faults = 1  a_s_mask = 2  a_s_risk = 16
```

```
.inputs 13
.outputs 4        PLA DIL
.products 20
```



```
0 1 X X X X X X 0 X 0 1 X      - 1 - -
0 1 X X X X X X 0 1 0 X 0      - 1 - -
0 1 X X X X X X 0 1 X 1 0      - 1 - -
1 0 X X X X X X 0 X 0 1 X      1 - - -
1 0 X X X X X X 0 1 0 X 0      1 - - -
1 0 X X X X X X 0 1 X 1 0      1 - - -
X X 0 1 X X X X 0 0 1 0 X      - 1 - -
X X 1 0 X X X X 0 0 1 0 X      1 - - -
X X X X 0 X X 0 0 0 1 1 X      - - - 1
X X X X 0 1 X 0 0 0 1 1 X      - 1 - -
X X X X 1 0 X 0 0 0 1 1 X      1 - - 1
X X X X 0 X 0 1 0 0 1 1 X      - - 1 -
X X X X 0 X 1 0 0 0 1 1 X      - - 1 -
X X X X 1 0 0 1 0 0 1 1 X      1 - 1 -
X X X X X 0 1 0 0 0 1 1 X      - - 1 -
X X X X 0 1 0 X 0 0 1 1 X      - 1 - -
X X X X 0 1 X X 0 1 1 0 X      - 1 - -
X X X X 0 1 X X 0 1 X 0 1      - 1 - -
X X X X 1 0 X X 0 1 1 0 X      1 - - -
X X X X 1 0 X X 0 1 X 0 1      1 - - -
```

Sh fault location

(row	input	bit line)
(4	11	0)
(5	10	1)
(18	12	0)
(19	10	0)
(1	11	0)
(2	10	1)
(9	6	1)
(15	7	1)
(16	12	0)
(17	10	0)
(12	5	1)
(14	4	1)

Sh_faults = 12 s_g_mask = 0 s_a_mask = 7 (Total mask = 7)
s_g_risk = 1040

App fault location

(row	output)
(9	3)
(12	3)
(14	3)

App_faults = 3 a_s_mask = 3 a_s_risk = 128

```
.inputs   4
.outputs  4        PLA. F2
.products 12

X 0 1 0    1---
0 X 1 0    1---
0 0 1 X    1---
1 0 X 0    --1-
1 X 0 0    --1-
1 0 0 X    --1-
0 1 X 0    -1--
X 1 0 0    -1--
0 1 0 X    -1--
0 0 X 1    ---1
X 0 0 1    ---1
0 X 0 1    ---1
```

```
Sh_fault location
( row , input . bit line )
==========================
( 11 , 1 . 1 )
( 10 . 0 , 1 )
( 9 . 2 . 1 )
( 5 . 3 . 1 )
( 4 . 1 . 1 )
( 3 . 2 . 1 )
( 8 , 3   1 )
( 7 . 0 . 1 )
( 6 , 2   1 )
( 2 . 3 , 1 )
( 1 . 1   1 )
( 0 . 0 . 1 )

Sh_faults = 12. s_g_mask = 0, s_a_mask = 12 (Total mask = 12)
                s_g_risk = 1
```

```
.inputs 9
.outputs 7          PLA MAS
.products 54

0 1 1 X 0 0 0 X 1      0 0 1 0 0 0 0
0 0 0 X 0 0 0 0 1      0 0 1 0 1 0 0
1 X X X 0 0 0 0 1      0 0 1 0 1 0 0
X X X X 0 0 0 1 0      0 0 1 0 1 0 0
X 0 X X 0 0 0 1 1      0 0 1 0 1 0 0
1 1 0 X 0 0 0 1 1      0 0 1 1 0 0 0
0 1 1 X 0 0 1 0 0      0 0 1 0 0 0 0
0 0 0 X 0 0 1 0 0      0 0 1 0 1 0 0
1 X X X 0 0 1 0 X      0 0 1 1 0 0 0
0 1 1 X 0 0 1 0 1      0 0 1 0 1 0 0
0 0 0 X 0 0 1 0 1      0 0 1 1 0 0 0
X X X X 0 0 1 1 0      0 0 1 1 0 0 0
X X X X 0 0 1 1 1      0 1 0 0 1 0 0
0 X X X 0 1 0 0 X      0 1 0 0 1 0 0
1 0 1 X 0 1 0 0 0      0 1 0 0 1 0 0
1 1 0 X 0 1 0 0 0      0 1 1 0 0 0 0
1 0 1 X 0 1 0 0 1      0 1 1 0 0 0 0
1 1 0 X 0 1 0 0 1      0 1 1 0 1 0 0
0 1 1 X 0 1 0 1 0      0 1 0 0 1 0 0
X 0 X X 0 1 0 1 X      0 1 1 0 0 0 0
1 1 0 X 0 1 0 1 X      0 1 1 0 1 0 0
0 1 1 X 0 1 0 1 1      0 1 1 0 0 0 0
X X X X 0 1 1 0 0      1 1 0 0 0 0 0
0 X X X 0 1 1 0 1      1 1 0 1 1 0 0
0 1 1 X 0 1 1 1 0      1 0 0 0 0 0 0
1 0 1 X 0 1 1 1 0      1 0 0 0 1 0 0
0 1 1 X 0 1 1 1 1      1 0 0 1 0 0 0
0 0 0 X 0 1 1 1 1      1 0 0 1 1 0 0
1 1 0 X 0 1 1 1 1      1 0 1 1 0 0 0
X X 1 0 1 0 0 0 1      0 0 0 1 0 0 0
X X 0 0 1 0 0 0 1      0 0 0 1 1 0 0
X X X 0 1 0 0 1 0      0 0 0 1 1 0 0
X X 1 0 1 0 0 1 1      0 0 0 1 1 0 0
X X 0 0 1 0 0 1 1      0 0 1 1 0 0 0
X X 1 0 1 0 1 0 X      0 0 1 1 0 1 0
X X 0 0 1 0 1 0 0      0 0 1 1 0 0 1
X X 0 0 1 0 1 0 1      0 1 0 0 0 0 1
X X 1 0 1 0 1 1 X      0 1 0 0 0 1 0
X X 0 0 1 0 1 1 0      0 1 0 0 0 0 1
X X 0 0 1 0 1 1 1      0 1 0 1 0 0 1
X X 1 0 1 1 0 0 0      0 1 0 1 0 1 0
X X 0 0 1 1 0 0 0      0 1 0 1 1 0 1
X X 1 0 1 1 0 0 1      0 1 0 1 1 0 0
X X 0 0 1 1 0 0 1      0 0 0 1 0 0 0
X X 1 0 1 1 0 1 0      0 0 0 1 1 0 0
X X 0 0 1 1 0 1 0      0 1 0 0 0 0 0
X X 1 0 1 1 0 1 1      0 1 0 1 1 1 0
X X 0 0 1 1 0 1 1      0 0 1 1 0 0 1
X X 1 0 1 1 1 0 0      0 0 1 1 1 1 0
X X 0 0 1 1 1 0 0      0 1 1 1 0 0 1
X X 1 0 1 1 1 0 1      0 0 0 0 0 1 0
X X 0 0 1 1 1 0 1      0 0 0 0 0 0 1
X X X 0 1 1 1 1 0      1 1 1 0 0 0 0
X X X 0 1 1 1 1 1      1 1 1 1 0 0 0
```

```
Gr_fault location
( row   input )
==============
( 24    8 )
( 24 .  7 )
( 26 .  7 )
( 27    7 )
( 52    8 )
( 14 .  0 )
( 15 .  8 )
( 15    7 )
( 16    7 )
( 17 .  7 )
( 18 .  7 )
( 21 .  1 )
( 36 .  7 )
( 38    8 )
( 42 .  7 )
( 45    7 )
( 45 .  6 )
( 1     7 )
( 1 .   0 )
( 4     8 )
( 6     8 )
( 6     7 )
( 6 .   0 )
( 10    0 )
( 33    5 )
( 35    5 )
( 29    7 )
( 29 .  6 )
( 29 .  5 )
( 29    2 )
( 32    8 )
( 32    5 )
( 43    8 )
( 43 .  7 )
( 43 .  5 )
( 43    2 )
( 44 .  8 )
( 44 .  5 )
( 50    8 )
( 50    5 )
( 51 .  8 )
( 51    5 )
```

Gr_faults = 42. g_s_mask = 140 g_s_risk = 65

```
.inputs 10
.outputs 11      PLA MID
.products 12

X X X 0 1 0 0 1 X X      0 1 1 0 0 0 1 0 0 0 1
X X X 0 1 1 0 1 X X      0 0 1 1 0 0 0 1 0 0 1
X X X 1 0 0 0 1 X X      0 1 1 0 0 0 0 1 0 0 1
X X X 1 0 1 0 1 X X      0 0 1 1 0 1 0 1 0 0 1
X X X 1 1 0 0 1 X X      0 1 1 0 0 1 0 1 0 0 1
X X X 1 1 1 0 1 X X      0 0 1 1 0 0 0 0 1 0 1
X X X 0 0 1 0 1 X X      1 0 1 0 0 0 0 0 0 1 0
1 X X X X X 0 0 1 1      1 0 0 0 0 1 0 0 0 0 0
X X X X X X 1 0 0 1      1 0 0 0 0 1 0 0 0 0 0
X X 0 0 0 0 0 1 X X      1 0 0 0 0 1 0 0 0 0 0
X 1 X X X X 0 0 1 1      1 0 0 0 1 0 0 0 0 0 0
X X 1 0 0 0 0 1 X X      1 0 1 0 1 0 0 0 0 0 0
```

```
Gr fault location
( row   input )
===============
( 2   4 )
Gr_faults = 1. g_s_mask = 40. g_s_risk = 0
```

```
.inputs 8
. outputs 7          PLA MISEX1
.products  32

0 1 1 1 X X X X      1 0 0 0 0 0 0
1 0 1 0 X X X X      1 0 0 0 0 0 0
0 1 0 X X X X X      0 1 0 0 0 0 0
0 0 1 1 X X X X      0 1 0 0 0 0 0
1 0 0 1 X X X X      0 1 0 0 0 0 0
0 0 1 X X 1 X X      0 1 0 0 0 0 0
0 X 0 0 X X 1 X      0 1 0 0 0 0 0
0 1 X 1 X X X X      0 0 1 0 0 0 0
1 0 0 1 X X X X      0 0 1 0 0 0 0
0 1 0 X 1 X X X      0 0 1 0 0 0 0
0 0 1 0 X 0 X X      0 0 1 0 0 0 0
0 0 0 0 X X 0 X      0 0 1 0 0 0 0
1 0 1 0 X X X X      0 0 0 1 0 0 0
X 0 1 0 X 0 X X      0 0 0 1 0 0 0
0 1 0 X X X X 0      0 0 0 1 0 0 0
0 1 0 0 0 X X X      0 0 0 1 0 0 0
0 1 0 X X X X X      0 0 0 0 1 0 0
0 X 1 1 X X X X      0 0 0 0 1 0 0
X 0 1 0 X X X X      0 0 0 0 1 0 0
0 X 0 0 X X X X      0 0 0 0 1 0 0
1 0 0 1 X X X X      0 0 0 0 1 0 0
0 1 0 X X X X X      0 0 0 0 0 1 0
0 X 1 1 X X X X      0 0 0 0 0 1 0
1 0 0 1 X X X X      0 0 0 0 0 1 0
1 0 1 0 X X X X      0 0 0 0 0 1 0
0 0 1 X X 1 X X      0 0 0 0 0 1 0
0 X 0 0 X X 1 X      0 0 0 0 0 1 0
0 1 X 1 X X X X      0 0 0 0 0 0 1
1 0 0 1 X X X X      0 0 0 0 0 0 1
1 0 1 0 X X X X      0 0 0 0 0 0 1
0 1 0 X 0 X X X      0 0 0 0 0 0 1
X 0 1 0 X 0 X X      0 0 0 0 0 0 1
```

App_fault location
(row , output)

(24 6)	(14 4)
(23 6)	(13 4)
(20 6)	(12 4)
(15 6)	(11 4)
(13 6)	(10 4)
(12 6)	(9 4)
(10 6)	(8 4)
(8 6)	(7 4)
(7 6)	(6 4)
(4 6)	(5 4)
(1 6)	(4 4)
(0 6)	(3 4)
(30 5)	(2 4)
(29 5)	(1 4)
(28 5)	(0 4)
(27 5)	(31 3)
(20 5)	(29 3)
(17 5)	(24 3)
(16 5)	(10 3)
(15 5)	(1 3)
(14 5)	(28 2)
(12 5)	(27 2)
(9 5)	(23 2)
(8 5)	(20 2)
(7 5)	(4 2)
(6 5)	(0 2)
(5 5)	(30 1)
(4 5)	(28 1)
(3 5)	(26 1)
(2 5)	(25 1)
(1 5)	(23 1)
(0 5)	(21 1)
(31 4)	(20 1)
(30 4)	(16 1)
(29 4)	(15 1)
(28 4)	(14 1)
(27 4)	(9 1)
(26 4)	(8 1)
(25 4)	(29 0)
(24 4)	(24 0)
(23 4)	(12 0)
(22 4)	
(21 4)	
(15 4)	

Sh_fault location
(row , input , bit line)

(31	0	0)
(30	3	0)
(29	5	1)
(26	1	0)
(25	3	0)
(19	1	0)
(16	3	1)
(15	7	1)
(13	0	0)
(12	5	1)
(9	3	0)
(6	1	0)
(5	3	0)
(3	5	0)

Sh_faults = 14 s_g_mask = 3 s_a_mask = 39 (Total mask = 42)
s_g_risk = 108

App_faults = 85 a_s_mask = 457 a_s_risk = 112

```
.inputs  5
.outputs 3      PLA RD53
.products 32

1 X 1 1 1      1--
1 1 X 1 1      1--
1 1 1 1 X      1--
1 1 1 X 1      1--
X 1 1 1 1      1--
0 1 X 0 1      --1
X 0 1 1 0      --1
0 0 1 X 1      --1
1 X 0 0 1      --1
1 X 1 0 0      --1
1 1 0 X 0      --1
0 1 1 X 0      --1
1 0 0 1 X      --1
0 X 0 1 1      --1
X 1 0 1 0      --1
X 0 1 0 1      --1
0 1 1 1 0      -1-
0 0 0 1 0      -1-
0 1 0 0 0      -1-
1 1 1 1 1      -1-
0 0 1 0 0      -1-
0 0 1 1 1      -1-
1 1 1 0 0      -1-
1 1 0 1 0      -1-
0 1 1 0 1      -1-
0 1 0 1 1      -1-
1 0 1 1 0      -1-
1 0 0 0 0      -1-
1 1 0 0 1      -1-
0 0 0 0 1      -1-
1 0 1 0 1      -1-
1 0 0 1 1      -1-
```

Sh_fault location
(row , input , bit line)

```
( 15 , 0 , 1 )
( 14 , 0 , 0 )
( 10 , 3 , 0 )
( 7 , 3 , 1 )
( 4 , 0 , 0 )
( 3 , 3 , 0 )
( 2 , 4 , 0 )
( 1 , 2 , 0 )
( 0 , 1 , 0 )
```

App_fault location
(row , output)

```
( 31 , 2 )
( 30 , 2 )
( 28 , 2 )
( 26 , 2 )
( 25 , 2 )
( 24 , 2 )
( 23 , 2 )
( 22 , 2 )
( 21 , 2 )
( 16 , 2 )
( 19 , 0 )
```

Sh_faults = 9, s_g_mask = 8, s_a_mask = 2 (Total mask = 10)
s_g_risk = 4

App_faults = 11, a_s_mask = 0, a_s_risk = 0

```
1 X X 1 1 X 1  --1
X 1 X 1 1 X 1  --1
1 1 X X 1 X 1  --1
1 1 X 1 1 X X  --1
1 1 X 1 X X 1  --1
1 1 X X 1 1 X  --1
X 1 X 1 1 1 X  --1
1 1 X X X 1 1  --1
1 1 X 1 X 1 X  --1
X 1 X X 1 1 1  --1
1 X X X 1 1 1  --1
1 X 1 1 X X 1  --1
X X X 1 1 1 1  --1
1 X X 1 1 1 X  --1
X 1 X 1 X 1 1  --1
1 1 1 X X 1 X  --1
X 1 1 X X 1 1  --1
X 1 1 X 1 1 X  --1
X X 1 1 X 1 1  --1
X 1 1 1 X 1 X  --1
1 X 1 X 1 1 X  --1
X X 1 1 1 1 X  --1
1 1 1 X X X 1  --1
X X 1 X 1 1 1  --1
X 1 1 1 1 X X  --1
X 1 1 X 1 X 1  --1
X X 1 1 1 X 1  --1
1 1 1 X 1 X X  --1
1 X 1 1 1 X X  --1
1 1 1 1 X X X  --1
1 X 1 1 X 1 X  --1
1 X X 1 X 1 1  --1
X 1 1 1 X. X 1  --1
1 X 1 X X 1 1  --1
1 X 1 X 1 X 1  --1
X 0 0 0 1 1 0  1--
1 0 0 X 1 0 0  1--
0 1 0 0 X 1 0  1--
1 0 0 1 0 X 0  1--
1 1 0 X 0 0 0  1--
0 0 0 X 0 1 1  1--
1 0 0 0 0 1 X  1--
0 1 0 X 0 0 1  1--
X 1 0 0 1 0 0  1--
1 0 X 0 0 0 1  1--
0 1 0 1 X 0 0  1--
0 0 1 1 0 X 0  1--
0 1 1 0 X 0 0  1--
X 0 0 1 0 0 1  1--
0 0 1 X 0 0 1  1--
1 0 1 X 0 0 0  1--
0 0 0 1 X 1 0  1--
1 X 1 1 1 1 1  1--
0 0 X 1 1 0 0  1--
0 0 1 0 X 1 0  1--
0 0 0 X 1 0 1  1--
X 0 1 0 1 0 0  1--
0 1 1 X 0 0 0  1--
X 1 1 0 0 0 0  1--
0 1 0 0 0 1 X  1--
1 1 1 1 X 1 1  1--
1 0 0 0 1 0 X  1--
X 0 1 0 0 1 0  1--
0 1 0 0 1 0 X  1--
0 0 1 0 1 0 X  1--

1 1 1 X 1 1 1  1--
1 1 X 1 1 1 1  1--
0 1 1 0 0 0 X  1--
0 X 1 0 0 1 0  1--
X 1 1 1 1 1 1  1--
0 1 0 X 0 1 0  1--
1 1 1 1 1 X 1  1--
0 0 1 0 0 1 X  1--
1 1 1 1 1 1 X  1--
X 1 0 0 0 1 0  1--
0 0 0 0 0 0 1  -1-
0 0 0 1 0 0 0  -1-
1 0 0 0 0 0 0  -1-
1 0 0 1 0 0 1  -1-
0 0 0 0 1 0 0  -1-
0 0 0 1 1 0 1  -1-
1 0 0 0 1 0 1  -1-
0 1 0 1 0 0 1  -1-
0 1 0 0 0 0 0  -1-
0 0 0 1 0 1 1  -1-
1 1 0 0 0 0 1  -1-
0 0 1 1 0 0 1  -1-
0 1 0 0 1 0 1  -1-
1 0 0 0 0 1 1  -1-
0 0 0 0 0 1 0  -1-
1 0 1 0 0 0 1  -1-
0 0 0 0 1 1 1  -1-
0 1 0 0 0 1 1  -1-
0 0 1 0 1 0 1  -1-
0 0 1 0 0 0 0  -1-
0 1 1 0 0 0 1  -1-
0 0 1 0 0 1 1  -1-
1 0 0 1 1 0 0  -1-
1 1 0 1 0 0 0  -1-
1 1 0 1 1 0 1  -1-
1 0 0 1 0 1 0  -1-
0 1 0 1 1 0 0  -1-
1 0 0 1 1 1 1  -1-
0 0 0 1 1 1 0  -1-
1 1 0 1 0 1 1  -1-
0 1 0 1. 1 1 1  -1-
1 0 1 1 1 0 1  -1-
1 1 0 0 1 0 0  -1-
0 1 0 1 0 1 0  -1-
1 0 0 0 1 1 0  -1-
1 0 1 1 0 0 0  -1-
1 1 0 0 1 1 1  -1-
0 0 1 1 1 0 0  -1-
1 1 0 0 0 1 0  -1-
1 1 1 1 0 0 1  -1-
1 0 1 0 1 0 0  -1-
0 1 1 1 0 0 0  -1-
1 1 1 0 0 0 0  -1-
1 0 1 1 0 1 1  -1-
0 1 0 0 1 1 0  -1-
0 1 1 1 1 0 1  -1-
1 1 1 0 1 0 1  -1-
0 0 1 1 0 1 0  -1-
0 1 1 0 1 0 0  -1-
0 0 1 1 1 1 1  -1-
1 0 1 0 0 1 0  -1-
0 1 1 1 0 1 1  -1-
0 0 1 0 1 1 0  -1-
1 0 1 0 1 1 1  -1-

0 1 1 0 0 1 0  -1-
1 1 1 0 0 1 1  -1-
0 1 1 0 1 1 1  -1-
1 1 0 1 1 1 0  -1-
1 1 1 1 1 0 0  -1-
1 0 1 1 1 1 0  -1-
1 1 1 1 0 1 0  -1-
0 1 1 1 1 1 0  -1-
1 1 1 0 1 1 0  -1-
1 1 1 1 1 1 1  -1-
```

SEE FAULT DATA -->

Sh_fault location (row, input, bit line)	App_fault location (row, output)
(34 5 , 0)	(12 , 0 , 0)	(140 2)
(34 , 3 0)	(11 , 5 , 0)	(139 2)
(34 , 1 0)	(11 , 4 0)	(138 2)
(33 4 , 0)	(11 , 1 0)	(137 , 2)
(33 , 3 , 0)	(10 , 3 , 0)	(136 , 2)
(33 , 1 , 0)	(10 , 2 0)	(135 , 2)
(32 , 5 , 0)	(10 1 0)	(134 2)
(32 , 4 , 0)	(9 3 , 0)	(133 2)
(32 , 0 , 0)	(9 , 2 , 0)	(132 , 2)
(31 , 4 , 0)	(9 , 0 0)	(130 , 2)
(31 , 2 , 0)	(8 , 6 , 0)	(128 , 2)
(31 , 1 , 0)	(8 , 4 , 0)	(126 , 2)
(30 , 6 , 0)	(8 , 2 , 0)	(123 , 2)
(30 , 4 0)	(7 , 4 , 0)	(122 , 2)
(30 , 1 0)	(7 , 3 , 0)	(120 , 2)
(29 , 6 0)	(7 , 2 , 0)	(116 2)
(29 5 0)	(6 , 6 , 0)	(113 , 2)
(29 , 4 , 0)	(6 , 2 , 0)	(108 , 2)
(28 , 6 , 0)	(6 , 0 , 0)	(107 2)
(28 , 5 , 0)	(5 , 6 , 0)	(106 , 2)
(28 1 0)	(5 , 3 , 0)	(104 2)
(27 , 6 , 0)	(5 , 2 , 0)	(101 , 2)
(27 , 5 , 0)	(4 , 5 , 0)	(75 , 2)
(27 , 3 , 0)	(4 , 4 , 0)	(73 , 2)
(26 , 5 0)	(4 , 2 , 0)	(71 , 2)
(26 1 , 0)	(3 , 6 , 0)	(68 , 2)
(26 , 0 , 0)	(3 , 5 , 0)	(67 , 2)
(25 , 5 , 0)	(3 , 2 , 0)	(60 , 2)
(25 , 3 0)	(2 , 5 , 0)	(52 , 2)
(25 , 0 , 0)	(2 , 3 , 0)	(140 , 0)
(24 , 6 , 0)	(2 , 2 , 0)	(131 , 0)
(24 , 5 , 0)	(1 , 5 , 0)	(129 0)
(24 , 0 , 0)	(1 , 2 , 0)	(127 , 0)
(23 , 3 , 0)	(1 , 0 , 0)	(125 , 0)
(23 , 1 0)	(0 , 5 , 0)	(124 , 0)
(23 , 0 , 0)	(0 , 2 , 0)	(121 , 0)
(22 , 5 , 0)	(0 , 1 , 0)	(119 0)
(22 , 4 , 0)	(76 , 0 , 1)	(118 0)
(22 , 3 , 0)	(75 , 6 0)	(117 , 0)
(21 , 6 , 0)	(74 6 , 1)	(115 , 0)
(21 , 1 0)	(73 , 5 , 0)	(114 0)
(21 , 0 , 0)	(72 , 3 , 1)	(112 0)
(20 , 6 , 0)	(71 , 0 , 0)	(111 0)
(20 , 3 , 0)	(70 , 1 , 1)	(110 , 0)
(20 , 1 , 0)	(69 , 6 1)	(109 , 0)
(19 6 , 0)	(68 2 , 0)	(105 , 0)
(19 , 4 , 0)	(67 , 3 , 0)	(103 0)
(19 0 , 0)	(66 , 6 , 1)	(102 , 0)
(18 4 , 0)	(65 , 6 , 1)	(100 0)
(18 1 , 0)	(64 , 6 , 1)	(99 , 0)
(18 , 0 0)	(63 , 6 , 1)	(98 0)
(17 , 6 , 0)	(62 0 , 1)	(97 , 0)
(17 , 3 0)	(61 6 , 1)	(95 , 0)
(17 , 0 , 0)	(60 , 4 , 0)	(94 , 0)
(16 , 4 , 0)	(59 6 , 1)	(93 0)
(16 , 3 , 0)	(58 , 0 , 1)	(92 , 0)
(16 , 0 0)	(57 3 , 1)	(90 , 0)
(15 6 , 0)	(56 , 0 , 1)	(89 0)
(15 4 , 0)	(54 , 4 , 1)	(88 0)
(15 3 , 0)	(52 , 1 0)	(87 , 0)
(14 , 4 0)	(47 , 4 , 1)	(86 , 0)
(14 , 2 0)	(43 , 0 , 1)	(84 0)
(14 , 0 , 0)	(39 3 , 1)	(83 0)
(13 , 6 , 0)	(37 , 4 , 1)	(82 0)
(13 , 2 0)	(36 , 3 , 1)	(80 0)
(13 , 1 , 0)	(35 , 0 , 1)	
(12 2 0)		

Sh_faults = 134 s_g_mask = 44, s_a_mask = 29 (Total mask = 73)
s_g_risk = 5

App_faults = 65 a_s_mask = 0 a_s_risk = 29

```
.inputs 10
.outputs 4          PLA_SA02
.products 58

X X 0 X 1 0 0 X X 0      ~ ~ 1 1
X X X X 0 0 1 X 0 0      ~ ~ 1 1
X 0 X X X 0 0 X 1 0      ~ ~ 1 1
0 X 1 X 0 0 X X X 0      ~ ~ 1 1
X 1 X 0 X 0 X X 0 0      ~ ~ 1 1
1 X 0 X X 0 X 0 X 0      ~ ~ 1 1
X X X 0 X X X 0 X 0      ~ ~ 1 ~
X 0 0 0 0 0 0 1 0 0      1 ~ ~ 1
1 1 1 1 0 0 0 1 0 0      ~ 1 ~ ~
1 1 0 1 0 0 0 1 1 0      ~ 1 ~ ~
1 0 1 1 1 0 0 1 0 0      ~ 1 ~ ~
0 1 0 1 0 0 1 1 1 0      ~ 1 ~ ~
1 1 1 1 1 0 0 1 1 0      ~ 1 ~ ~
1 1 1 1 0 0 1 1 1 0      ~ 1 ~ ~
1 1 1 1 1 0 1 1 0 0      ~ 1 ~ ~
1 1 0 1 1 0 1 1 1 0      ~ 1 ~ ~
1 0 1 1 1 0 1 1 1 0      ~ 1 ~ ~
1 0 1 0 0 0 0 1 0 0      ~ 1 ~ 1
0 1 1 1 1 0 1 1 1 0      ~ 1 ~ ~
0 1 0 1 1 0 1 0 1 0      ~ 1 ~ ~
1 1 1 1 1 0 1 0 1 0      ~ 1 ~ ~
1 0 1 0 1 0 1 1 0 0      ~ 1 ~ ~
1 1 1 0 1 0 1 1 1 0      ~ 1 ~ ~
0 1 X 1 1 0 1 0 1 0      1 ~ ~ ~
1 0 1 0 1 0 1 1 X 0      1 ~ ~ ~
1 0 0 1 0 0 0 1 0 0      ~ 1 ~ ~
0 1 0 1 0 0 0 1 0 0      ~ 1 ~ ~
0 1 0 1 0 0 0 0 1 0      ~ 1 ~ ~
0 X 0 1 0 0 0 0 0 0      1 ~ ~ ~
0 0 0 1 0 0 0 0 0 0      ~ 1 ~ ~
0 0 0 0 0 0 0 1 0 0      ~ 1 ~ ~
X X X X X 1 X X X 0      ~ ~ 1 ~
0 0 0 1 0 0 0 X 0 0      1 ~ ~ ~
1 0 1 1 1 0 X 1 0 0      1 ~ ~ ~
1 1 1 1 X 0 0 1 0 0      1 ~ ~ ~
X X X 0 1 0 0 X X 0      ~ ~ ~ 1
0 1 0 1 1 0 1 X 1 0      1 ~ ~ ~
X 1 0 1 0 0 1 1 1 0      1 ~ ~ ~
1 1 X 1 0 0 0 1 1 0      1 ~ ~ ~
X X 0 1 X 0 X X 0 0      ~ ~ ~ 1
X X X 1 0 0 X 0 X 0      ~ ~ ~ 1
0 X X 1 X 0 0 X X 0      ~ ~ ~ 1
X 1 X 0 X 0 X 0 X 0      ~ ~ ~ 1
X 0 X 1 0 0 X X X 0      ~ ~ ~ 1
X X 1 X 0 0 X 0 X 0      ~ ~ 1 1
0 X X X 1 X 0 X X 0      ~ ~ 1 ~
X 0 X X X 0 X 0 1 0      ~ ~ 1 1
0 1 X 0 X 0 X X X 0      ~ ~ 1 1
X X 0 X X 0 1 X 0 0      ~ ~ 1 1
X X X X X 0 1 0 0 0      ~ ~ 1 1
X 1 0 0 X 0 X X X 0      ~ ~ 1 1
0 0 X X X 0 X X 1 0      ~ ~ 1 1
X 0 X X 0 0 X X 1 0      ~ ~ 1 1
X X X X 1 0 0 0 X 0      ~ ~ 1 1
X 0 0 X X 0 X X 1 0      ~ ~ 1 1
0 X X X X 0 1 X 0 0      ~ ~ 1 1
X 1 X 0 0 0 X X X 0      ~ ~ 1 1
X 1 X 0 X X 0 X X 0      ~ ~ 1 ~
```

```
SH_fault location
( row   input   bit line )
==========================
( 23 . 2 . 1 )
( 28 : 1 . 1 )
( 32 . 7 : 1 )
( 36 : 7 , 1 )
( 0 . 0 . 1 )
( 0 , 7 . 1 )
( 1 . 0 : 1 )
( 1   2 . 1 )
( 1 . 7 , 1 )
( 2 . 0 . 1 )
( 2 . 2 . 1 )
( 2 . 4 . 1 )
( 2 . 7 , 1 )
( 3 . 7   1 )
( 4 : 0 , 1 )
( 4 . 2   1 )
( 4 , 4   1 )
( 4 , 6 . 1 )
( 4   7 . 1 )
( 6 . 5 , 0 )
( 44 . 0 . 1 )
( 45 . 2 . 1 )
( 45 . 5 : 0 )
( 45 , 7 : 1 )
( 46 , 0 : 1 )
( 46 , 2 : 1 )
( 46   4 : 1 )
( 46 , 6 . 1 )
( 47 . 2 . 1 )
( 47   4 . 1 )
( 47 . 6   1 )
( 47   7 : 1 )
( 47 , 8 , 1 )
( 48 , 0   1 )
( 48 : 4 , 1 )
( 48   7 . 1 )
( 49 . 0   1 )
( 49 . 2   1 )
( 49 . 4   1 )
( 50 . 0 . 1 )
( 50 . 4   1 )
( 50 . 6 , 1 )
( 50 , 7 . 1 )
( 50   8   1 )
( 51 . 2 . 1 )
( 51 . 4   1 )
( 51 , 6   1 )
( 51 . 7 : 1 )
( 52 : 0 . 1 )
( 52 , 2   1 )
( 52 , 6   1 )
( 52 . 7   1 )
( 53   0 . 1 )
( 53 , 2   1 )
( 53 , 3   1 )
( 54   0   1 )
( 54 . 4   1 )
( 54 , 6 : 1 )
( 54   7   1 )
( 55 : 2   1 )
( 55 : 4   1 )
( 56   7 , 1 )
( 56 . 8 , 1 )
( 57 , 0   1 )
( 57   2   1 )
( 57 : 4 . 1 )
( 57 , 5 , 0 )
( 57 , 7 , 1 )
( 57 . 8 , 1 )
( 35 , 2 . 1 )
( 35 . 7 . 1 )
( 39   0 , 1 )
( 39 . 1   1 )
( 39 : 4 , 0 )
( 39 : 6 , 0 )
( 39 : 7 , 1 )
( 40 , 0 : 0 )
( 40 , 1 . 1 )
( 40 : 2 , 0 )
( 40 , 6 : 1 )
( 40 , 8 , 1 )
( 41 . 7 . 1 )
( 42 , 0 . 1 )
( 42 . 2 . 1 )
( 42 , 4 , 1 )
( 42 : 6 . 1 )
( 42 , 8 , 1 )
( 43 , 0 . 1 )
( 43 , 2   1 )
( 43 , 6 . 0 )
( 43 : 7 , 1 )
( 43 : 8 : 0 )
```

```
App_fault locatio
( row   output )
================
( 8 , 0 )
( 9   0 )
( 10   0 )
( 11   0 )
( 19 . 0 )
( 21 . 0 )
( 29 . 0 )
( 30   0 )
( 42 . 2 )
( 25 . 3 )
( 26   3 )
( 27 : 3 )
( 28   3 )
( 29 , 3 )
( 30 : 3 )
( 32 . 3 )
```

Sh_faults = 95 s_g_mask = 91 s_a_mask = 9 (Total mask = 100)
s_g risk = 514

App_faults = 16 a_s_mask = 4 a_s_risk = 38

```
.inputs    10
.outputs   3       PLA_SR
.products  15

X X X X X X 1 0 1 1      0 0 1
X X X X X 1 1 1 1 0      0 1 0
X X 1 1 1 X 1 1 0 0      0 1 0
0 X X 1 1 X 1 1 0 0      0 1 0
0 1 X X 1 X 1 1 0 0      0 1 0
X 0 1 X 1 X 1 1 0 0      0 1 0
0 1 X X X 1 1 1 1 X      0 1 0
1 0 X X X 1 1 1 1 X      0 1 0
X X X X X 0 1 1 1 0      1 0 0
X X 1 1 0 X 1 1 0 0      1 0 0
0 X X 1 0 X 1 1 0 0      1 0 0
0 1 X X 0 X 1 1 0 0      1 0 0
X 0 1 X 0 X 1 1 0 0      1 0 0
0 1 X X X 0 1 1 1 X      1 0 0
1 0 X X X 0 1 1 1 X      1 0 0
```

```
Sh_fault location
( row , input , bit line )
==========================
( 7 , 9 , 1 )
( 6 , 9 , 1 )
( 5 , 3 , 0 )
( 4 , 3 , 0 )
( 3 , 2 , 0 )
( 3 , 1 , 0 )
( 2 , 1 , 1 )
( 2 , 0 , 1 )
( 14 , 9 , 1 )
( 13 , 9 , 1 )
( 12 , 3 , 0 )
( 11 , 3 , 0 )
( 10 , 2 , 0 )
( 10 , 1 , 0 )
( 9 , 1 , 1 )
( 9 , 0 , 1 )

Sh_faults = 16, s_g mask = 0. s_a_mask = 0 (Total mask = 0)
                s_g_risk = 180
```