Conditional Stuck-At Fault Model for PLA Test Generation

Olivian E. Cornelia

Department of Electrical Engineering

McGill University

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

December, 1987

Abstract

This thesis presents a new fault model called the conditional stuck-at (CSA) fault model. In this model, each stuck-at fault on a line might have associated with it a condition which requires that a test pattern for this fault should, in addition to testing for the fault, produce a specific binary value on another specified line in the gate model. The new fault model can efficiently model defects which do not result in purely stuck-at faults, such as bridging faults, and missing or additional transistors faults. This allows deterministic test pattern generation not only for stuck-at faults, but also for bridging faults and for additional or missing transistor faults. The conditional stuck-at fault model has been applied to Programmable Logic Arrays and it has been shown that all single stuck-at faults, single cross-point, and single bridging faults in a PLA can be covered by this new model. The CSA fault model allows one to solve the problem of test generation for PLA's by using the classical test pattern generation and simulation tools after minor changes, without requiring an extended gate model of the PLA. Since all the three most likely to occur faults in PLA's are treated under a unique fault model, the CSA fault model enables one to test embedded PLA's without requiring any additional controllability/observability schemes or Built-In Self Test. The viability of the conditional stuck-at fault model is demonstrated by simulation results for some benchmarking PLA's. Coverage figures are reported for stuck-at faults, for cross-point faults, and for bridging faults.

Résumé

Cette thèse propose un nouveau modèle de défaut appelé "défaut bloqué conditionnel" (DBC). Avec le modèle DBC, chaque défaut "bloqué" d'une ligne d'un circuit peut dans certains cas être associé à une condition sur une autre ligne. Cette condition exige qu'un vecteur de test pour le défaut "bloqué" doit, en plus de détecter le défaut "bloqué", produire une certaine valeur binaire sur l'autre ligne specifiée. Ce nouveau modéle de défaut peut servir à modeliser de manière éfficace, des types de défauts qui ne peuvent être modelisés par le modèle de défaut "bloqué". Des exemples de ces types de défauts sont: "point de traverse", "de pont". Ceci permet la génération algorithmique de vecteurs de test non seulement pour des défauts de type "bloqué", mais aussi pour des défauts de type "point de traverse" et "de pont". Le modèle DBC a été appliqué aux Structures de Logique Programmables (SLPs). Il a été demontré que tous les défauts simples de type "bloqué", "point de traverse", ou "de pont"d'une SLP, peuvent être détecté par le modèle DBC. En vue de résoudre le problème de la génération des vecteurs de test pour une SLP, le modèle de défaut DBC permet l'utilisation des outils classiques de génération de vecteurs de test et de simulation de défauts auxquels doivent être apportés que de simples modifications, et donc ne necessite pas que la SLP soit décrite par un modèle de portes logiques spécialement élaboré a cet éffet. Puisque les trois types de défauts les plus courants dans une SLP peuvent être traité par l'unique modèle DBC, ce dernier permet donc la vérification de SLPs intégrées dans un circuit logique sans avoir recours à des moyens spéciaux de contrôle et d'obseravation, ou à une structure d'autovérification. L'attrait du modèle DBC est soutenu par des résultats de simulations obtenus sur des SLPs qui servent d'exemples pour la comparaison de différents algorithmes. Des pourcentages de détection de défauts "bloqué", "point de traverse" et "de pont" sont inclus dans les résultats.

Acknowledgements

It is a great pleasure to express my sincere appreciation to Professor Vinod K. Agarwal for his supervision and guidance of this research project. With tremendous patience and competence, he provided the encouragement needed in the hard times of this work, and always shared the enthusiasm when results were obtained. Special thanks are due to Professor Nicholas Rumin and to Professor Janusz Rajski for helpful technical discussions.

Thanks are also due to the Natural Sciences and Engineering Research Council of Canada for providing financial assistance for this research project.

I sincerely thank my colleague, André Ivanov for his assistance in understanding the details of a large ATPG and simulation tool. Thanks are also due to all the students in the VLSI Group for creating such a pleasant work environment.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Advances in microelectronic technology allow hundreds of thousands of transistors to be placed on a single chip. Designing such a very complex integrated circuit represents a serious challenge and requires a structured approach as well as very efficient computer-aided design tools. To cope with the design complexity, it has become necessary to use standard cells, automatic layout generation from boolean equations, and, for specific applications, automatic layout generation from a high level description of the circuit (silicon compilation). Along these lines, array structures have emerged as a very efficient way to implement logic blocks which may be used afterwards as basic components. One of the most popular type of arrays are the Programmable Logic Arrays (PLA's). Their regularity allows automatic layout generation, which decreases significantly the design time, especially in the case of design iterations.

PLA's are widely used to implement two-level multiple output boolean functions. The PLA's used as standard parts are usually programmed in the field by blowing fuses (FPLA's). PLA's may also appear as blocks in larger integrated circuits or as standard parts. PLA's can be implemented both in bipolar and in MOS technologies. When used in larger integrated circuits, the PLA's are called embedded PLA's. They are frequently used to implement instruction decoders and, along with memory elements, finite state machines [1], [2].

A PLA, as shown in figure 1.1, basically consists of two rectangular adjacent arrays of wires representing the two levels of the logic functions they implement. Traditionally, the two arrays are called AND and OR, due to the function they implement

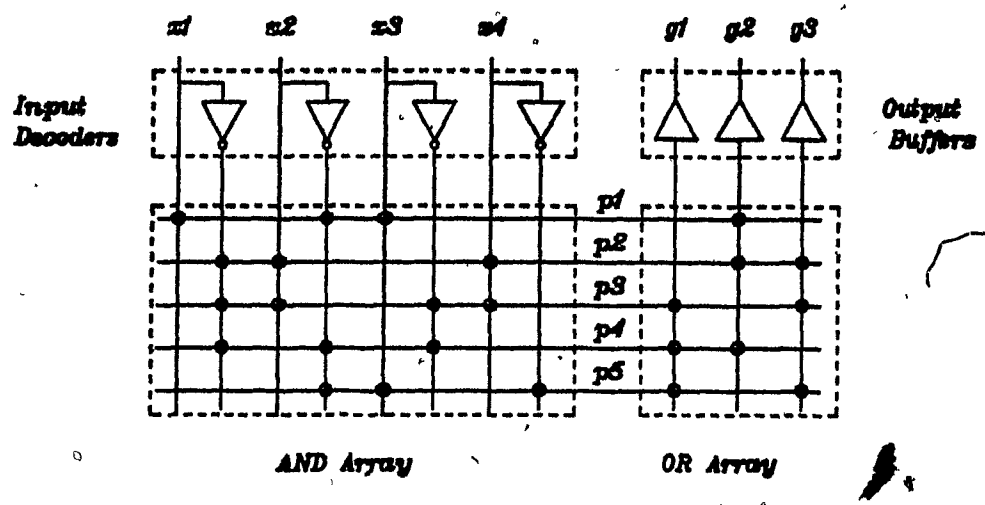
in bipolar technology. A NOR-NOR implementation is commonly used in the MOS technology, but the names for the arrays are still preserved. The vertical lines in the AND array are the decoded values of the inputs and are called *bit lines*. Usually, each input is decoded separately into two bit lines, as shown in figure 1.1. It is possible however to decode two input lines into four bit lines [3]. It has been shown in [4] that this implementation is more efficient as far as occupied area is concerned. In this thesis, only the single input decoder implementation will be analyzed.

The horizontal lines in both arrays are called *product lines*. In the AND array, each product line is an output of a gate, either AND or NOR, depending on the implementation. In the OR array, product lines are input lines for the gates in the array (fig. 1a). The vertical lines in the OR array represent the OR gates and hence the outputs of the PLA. For the NOR-NOR implementation, the second level of NOR gates is followed by inverters and therefore the NOR gates in the OR plane plus the output inverters implement on OR function.

The intersection of a horizontal and a vertical line is called a cross-point. Here, a device (transistor or diode) may or may not be present. If a device exists, then the product/output line depends on the bit/product line, respectively. Usually, such a cross-point is called a used cross-point. If no device is present, then no dependence exists between the two lines. In the symbolic PLA representation shown in figure 1.1*a*, a device is represented by a dot at the corresponding cross-point.

A PLA can be very well described in the cubical notation. The representation of the PLA in cubical notation, called personality matrix, consists of two adjacent two-dimensional arrays of symbols corresponding to the cross-points in the AND array and OR array. For the AND array, each entry in the personality matrix represents a pair of cross-points, at the intersection of two bit lines (of the same input) and some product line. If neither cross-point is used, then the entry is "x" (or "–"). If a device is placed on the inverted input, then the symbol is "0". If a device is placed on the non-inverted input. then the symbol is "1". In the OR array, if a device exists at some cross-point, then the entry in the personality matrix is "1", otherwise it is "0". The cubical

a) Symbolic Representation :



b) Cubical Representation :

1 0 1 x	0 1 0
0 1 x 1	0 1 1
0 1 0 1	1 0 1
0 0 0 x	1 1 0
x 0 1 0	1 0 1

c) Logic Functions :

$$g_1 = \overline{x_1}\,x_2\overline{x_3}\,x_4 + \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_2}\,x_3\,\overline{x_4}$$

$$g_2 = x_1\overline{x_2}x_3 + \overline{x_1}\,x_2x_4 + \overline{x_1}\,\overline{x_2}\overline{x_3}$$

$$g_3 = \overline{x_1}\,x_2x_4 + \overline{x_1}\,x_2\overline{x_3}x_4 + \overline{x_2}x_3\,\overline{x_4}$$

d) NMOS NOR-NOR Implementation :

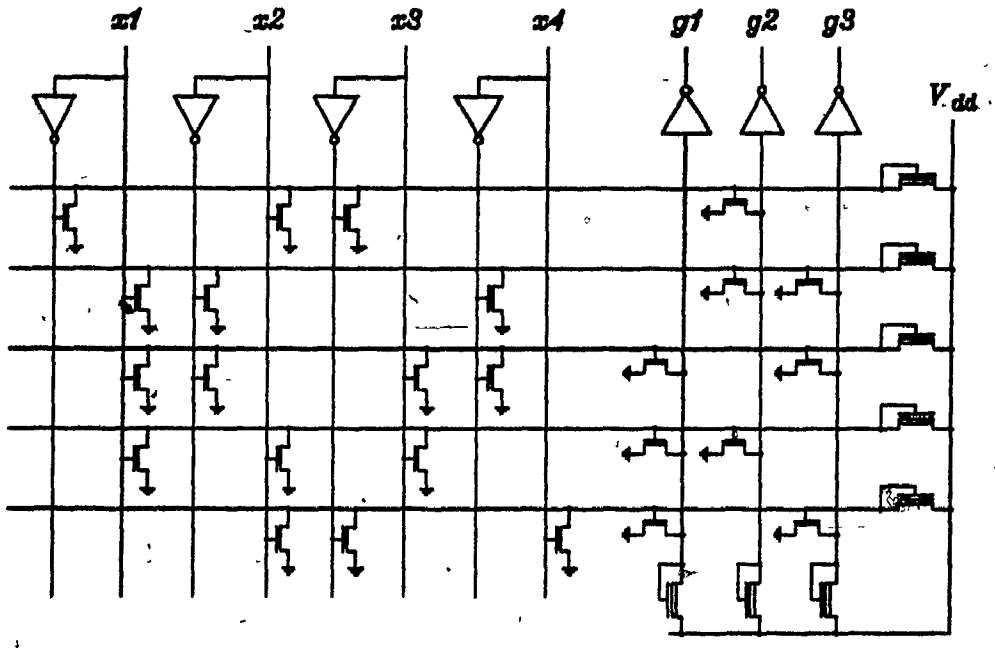


Figure 1.1 PLA Representations

representation of the PLA is widely used in logic minimization, test generation, and layout generation algorithms.

Due to their distinctive features, the PLA testing problem has become an important research problem [5],[6],[7],[8],[9],[10],[11],[12]. In general, the faults which are most likely to occur in PLA's are stuck-at faults, bridging faults, and cross-point faults. The layout information contained in the personality matrix and the regularity of the PLAs allow for more accurate modeling of defects. For the field programmable logic arrays, the defects which are the most likely to occur are the cross-point defects. They consist of some connection at a cross-point where there should not exist one ; or no connection at a cross-point where there should exist one. Clearly, programming errors can be mapped into cross-point defects. These defects have been modeled by the PLA-specific cross-point fault model. The cross-point fault model is considered to be superior to the stuck-at fault model because the functional effects of all single stuck-at faults except outputs stuck-at one can be mapped into functional effects of cross-point faults, while the reverse is not true unless extra circuitry is added [11],[13].

PLA bridging faults are particularly important due to the high density of the PLA layout which contains long wires running in parallel in both orthogonal directions. A bridging fault is caused by a short between two or more wires, none of which is ground or power supply. Usually, the PLA layout preserves the relative position of the lines in the PLA personality matrix. This provides the basic information required for considering bridging faults, that is the lines which may be involved in a bridge. The statistical analysis presented in [12] shows that the probability of occurrence for bridging faults is in the same range as the probability for stuck-at faults. Therefore, the problem of detecting bridging faults requires more research work. One of the important problems related to bridging faults is to determine what the effect of the bridge is. Traditionally, it has been assumed that a bridging fault has a definite logic effect, either AND or OR [7]. Under this assumption, it has been shown in [7] that single bridging faults in PLA's can be modeled as multiple cross-point faults. However, no exact reports exist on how well this kind of modeling performs in covering bridges. The analysis of PLA bridging faults under both AND and OR effects presented in [7] concludes by dividing

their coverage into three classes : fault detection guaranteed (by a single cross-point test set), not guaranteed, and detected if any testable cross-point is altered. Another problem related to bridging faults is the lack of an efficient way to model these faults. In [7], bridging faults are modeled as multiple cross-point faults. In [11], they are modeled as stuck-at faults by adding extra gates and the coverage of bridges is given at the computational cost of handling an extended model of the PLA.

Various methods for PLA testing have been proposed in the literature. Some representative approaches will be reviewed in chapter 2. Most of the deterministic test pattern generation algorithms for PLA's are based on the cross-point fault model [6],[9] and assume that the PLA inputs and outputs are directly accessible. These algorithms are designed for two-level circuits and the computation of test vectors relies on cube processing. Many heuristics have been developed to make these algorithms time efficient and generate compact test sets [7],[9]. Clearly, by using a specific fault model, these algorithms are applicable to PLA's only. For the case of embedded PLA's, where the assumption of accessible inputs and outputs does not hold, the tools based on the cross-point fault model are of limited value, unless additional controllability/observability schemes are used.

To avoid the computational effort related to test pattern generation, some easily testable PLA designs have been proposed in the literature [14],[15],[16]. The easy testability is achieved by adding extra hardware which increases the controllability and observability of all PLA crosspoints and makes the redundant cross-points testable. Quite often, easily testable PLA designs can be easily transformed into built-in self test designs [14], [17].

For embedded PLA's, the two alternatives proposed so far are built in self test [18],[17],[19] and enhanced controllability/observability schemes. Built-in self test for PLA's basically consists of on-chip test pattern generation and PLA output response analysis and may require further extra hardware for the PLA itself [20]. The second method involves off chip deterministic test pattern generation followed by test pattern application and output response observation. In order to control the PLA inputs (i.e. apply the test patterns) and observe the outputs, some extra circuitry has to be added.

It usually consists of a scan path [21],[22] which not only occupies extra silicon area but implies long test pattern application time. The PLA has to be treated separately since different fault models are used for the larger circuit and for the embedded PLA. Both methods for embedded PLA testing require extra design effort, extra hardware, may require extra pins, and may result in PLA performance degradation.

This thesis will introduce a new alternative for PLA testing, which is based on a new fault model called conditional stuck-at (CSA) fault model. It will be shown that stuck-at, cross-point, and bridging faults can be modeled as conditional stuck-at faults in the two-level logic model of the PLA without requiring extra gates or lines. The new fault model is applicable to general combinational circuits, as well as PLA's. Therefore, embedded PLA's can be tested as part of the larger circuit using a unique fault model and covering all PLA faults. This will avoid additional controllability/observability or BIST schemes, or an extended gate model for the PLA, which have been used so far for embedded PLA testing. Moreover, classical test pattern generation and simulation tools can be used after minor changes for conditional stuck-at faults. The use of the CSA fault model allows deterministic test pattern generation for stuck-at faults, cross-point faults, and for bridging faults. Therefore, exact coverage figures can be obtained for all these types of faults. Experiments and results will be presented, showing the viability of the new fault model.

The thesis is organized as follows. A review of the methods proposed in the literature for PLA testing is presented in chapter 2. Chapter 3 presents the new fault model and proves its capability to model stuck-at, cross-point, and bridging faults. Chapter 4 presents an analysis of the five types of bridging faults which may occur in PLA's and their mapping into the new fault model. Chapter 5 describes the experiments performed and the results obtained for a representative set of benchmarking PLA's. Concluding remarks are included in chapter 6.

Chapter 2 Literature Review

Digital circuit testing becomes increasingly important as digital systems become more complex. Therefore, there is much activity in trying to develop and improve testing techniques and test generation techniques. Various algorithms have been proposed for generating test vectors for combinational circuits [23],[24],[25], and many tools have been implemented based on these algorithms. These tools are widely used in the industry.

There are, however, certain types of circuits, such as PLA's, which cannot be handled directly by these general test pattern generation tools, and require specific test generation techniques. The PLA testing problem has received considerable attention due to the extensive use of PLA's, both as standard parts and as building blocks. Starting with the assumption that that PLA inputs and outputs are directly accessible, various deterministic test pattern generation and simulation algorithms have been proposed in the literature [6],[7],[9],[8], for PLA's only. Embedded PLA's, where inputs and outputs are not directly accessible, have to be provided with additional controllability/observability, or BIST schemes, or all PLA faults have to be modeled as stuck-at faults in an extended model of the PLA.

This chapter will review some of the representative approaches for general digital circuit test generation, PLA deterministic test pattern generation, and for embedded PLA testing.

2.1 Test Generation for Digital Integrated Circuits

The fault model which has received the most attention and has been the most widely used for integrated circuit testing is formulated at the gate level and is called the *stuck-at* fault model. This model assumes that physical failures cause input or output lines of logic gates to be permanently stuck at the logic 0 or 1 levels. For test generation purposes, it is possible to consider all possible multiple faults in a circuit, or to restrict the fault set to all possible single faults. Because of the exponential size of the set of multiple faults (3^n-1, in a circuit with n lines), the single stuck-at fault model is the most commonly used. This is also supported by theoretical results presented in [26] and a case study reported in [27] which shows that single fault test sets do provide high multiple fault coverage.

The generation of test patterns can be done in two different ways: randomly or deterministically. Deterministic test pattern generation algorithms can be further divided into two basic groups [28] : algebraic algorithms and structural algorithms. In algebraic algorithms, test generation is done by manipulating Boolean algebraic representations of the fault-free and faulty circuits. These algorithms are known to be very costly in terms of computation time and memory requirements. Structural algorithms use the topological representation of the circuit to generate test vectors, most often at the gate level. Several structural algorithms have been developed and successfully used in the industry. The most popular structural algorithms are the D-algorithm [23], PODEM [24], and FAN [25]. The PODEM algorithm will be briefly reviewed in this section.

PODEM is described by its author as an "implicit enumeration algorithm". Such algorithms are a subset of the branch and bound algorithms designed specifically for searching an n-dimensional 0-1 state space [24]. PODEM is based on a five-valued logic $\{0, 1, X, D, \overline{D}\}$, where $D(\overline{D})$ designates a value of 0(1) in the case of a faulty circuit and 1(0) in the case of the fault-free circuit.

The basic step of the algorithm essentially consists of assigning logic values to selected primary inputs. An implication is performed after each assignment to verify

the effects of the assignment on the propagation of the D or $\overline{D}$ values to some primary output. If no inconsistency occurs, another primary input is selected and a value is assigned to it. This is the branching operation. If an inconsistency is detected, then branching stops and bounding begins. There are two cases which require bounding :

1) The line value for the faulty line is the same as the stuck value. This implies that the fault is not activated under the current assignments.

2) There is no path from an internal line to a primary output such that the internal line is at the D or $\overline{D}$ value, and all other lines on the path still have unassigned values. That is, propagation of the effect of the fault is impossible under the current assignments.

Bounding consists of assigning the complementary value to the most recently assigned primary input. Branching resumes from this primary input with the newly assigned value. If both alternatives for some primary input have been tried unsuccessfully, then the branching resumes from the most recent primary input assignment where the alternative value has not yet been tried. This process continues until a test is found or the space is exhausted, that is, the fault is found to be untestable.

PODEM is best represented by a binary decision tree, as shown in figure 2.1, where the nodes represent assignments of the primary inputs. In this representation, branching consists of going as deeply as possible into the tree. Bounding consists of backtracking to the brother of the most recent ancestor node which still has an untried alternative. This is where branching resumes.

In practice, this binary decision tree can be implemented as a last-in first-out (LIFO) stack. Each initial primary input assignment results in pushing an unflagged node into the stack. Bounding results in popping nodes from the stack until an unflagged node is popped out. The search resumes by pushing this last node back into the stack, this time with the complementary value, as a flagged node. The process continues until a test is found, or until the stack becomes empty, which means that the fault is undetectable.

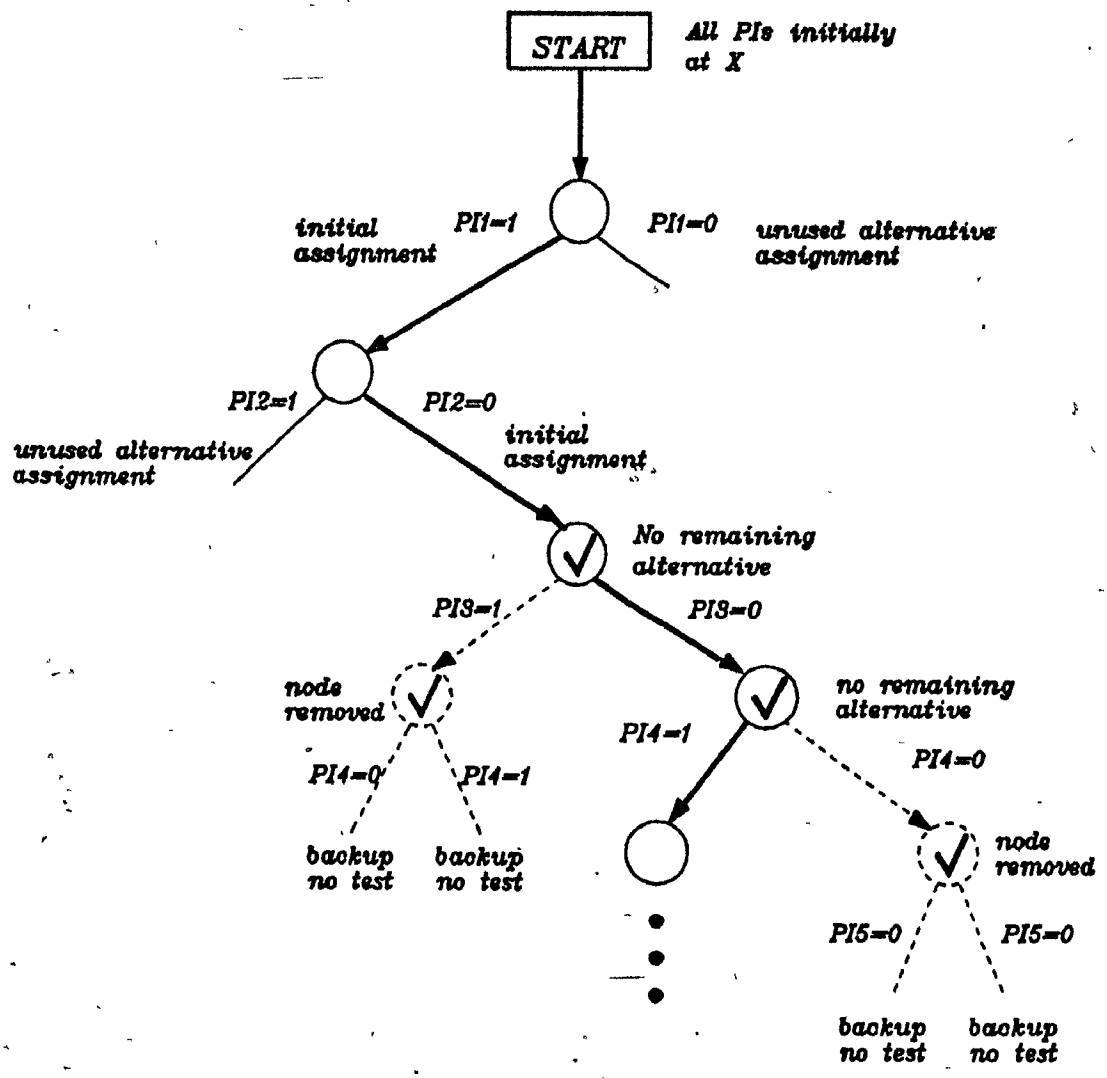


Figure 2.1 PODEM Decision Tree

An important problem is to decide which is the next primary input to be assigned a binary value. PODEM uses heuristics to decide on which PI the next assignment to be made, and what value should be assigned to it. For each fault, the search of a test vector is executed by repeatedly defining an objective. Specifically, an objective is defined as a pair (l, v), where l represents a line and v represents the desired logic level on line l. Given an objective, the choices made in the execution of the algorithm are such that they should help towards meeting the objective.

Initially, when the output of the faulty gate does not have a D or a $\overline{D}$ value, the initial objective is directed towards promoting setup for the gate. Once this setup

is obtained, the objective is aimed at propagating a D or a $\overline{D}$ one gate level closer to a primary output. Given the objective (l, v), the next step is to trace a path from line l to an unassigned primary input. Various heuristics are proposed in [24] for the backtrace operation. The backtrace operation leads eventually to a primary input and a binary value for it.

PODEM [24], D-algorithm [23], and FAN [25] are designed to generate test vectors for stuck-at faults. Therefore, these algorithms are of limited value for defects which cannot be modeled as stuck-at faults in the gate model of the circuit. Examples of such defects are additional transistor faults, or bridging faults, which may occur in array structured circuits.

2.2 PLA Faults

Generally, the faults which are most likely to occur in PLA's are stuck-at, cross-point, and bridging faults. The faults which are considered to be PLA specific are the cross-point faults, and a fault model has been introduced in [6] to model these faults. The fault model assumes that due to some defect, a transistor may disappear from a used cross-point or a transistor may appear at a unused cross-point. These two types of faults are usually called missing cross-point and additional cross-point fault, respectively. It has been shown in [6],[7] that PLA stuck-at faults, except output lines stuck-at one, can be modeled as cross-point faults.

The likelihood of occurrence of cross-point faults depends on the method of programming and on the technology used. In the case of field programmable PLA's, where programming is usually done by blowing fuses [6], the cross-point faults may occur due to fabrication defects or to programming errors. This was one of the main reasons for introducing a PLA specific fault model in [6]. For the case of programming at the mask level, the likelihood of occurrence of additional cross-point faults and missing cross-point faults is technology dependent. For the single metal layer MOS technology, the additional cross-point faults are less likely to occur. In contrast, for the double metal layer MOS technology where programming may be done by simply creating or

not creating contacts, the likelihood of occurrence of additional cross-point faults is comparable to that of missing cross-point faults.

Another important class of defects which may occur in integrated circuits in general are the short-circuits (shorts). For PLA's, they require increased attention due to the high density of the layout. Clearly, if one of the lines involved is a power supply line or a ground line, then the defect is accurately modeled by a stuck-at fault. If none of the lines involved is power or ground, then the short cannot be modeled directly as a stuck-at fault and it is called bridging fault. Bridging faults are caused by unwanted spots of conductive material (metal, polysilicon, or diffusion), or by missing insulator (silicon dioxide), or by a combination of both. For the particular case of PLA layout, characterized by a high density and long wires running in parallel in both orthogonal directions, bridging faults have an increased probability of occurrence. A probabilistic analysis of defects was presented in [12]. The analysis requires layout information and statistical information from the technological process. The results of the analysis applied to some example layout shows that the probability of occurrence for PLA same layer bridging faults is in the same range as the probability of occurrence for stuck-at faults.

A detailed analysis of bridging faults presented in [7] shows that bridging faults can be modeled as multiple cross-point faults, under the assumption that bridging faults have a certain logic effect, either AND or OR. Depending on their detectability, PLA bridging faults are divided into three classes [7] : fault detection guaranteed, fault detected if any cross-point in the set which models the bridge is testable, and fault detection not guaranteed. The results presented in [7] show, for each logic effect, which of the five types of PLA bridging faults belongs to each class. However, no exact reports exist on how well this kind of modeling performs in covering bridges. Most of the algorithms and tools available for PLA's deal with cross-point and stuck-at faults only.

2.3 Deterministic TPG Algorithms for PLA's

The deterministic TPG algorithms based on the cross-point fault model use

cube processing to compute test vectors. These computations are based on the effects of cross-point faults on the two-level functions implemented by PLA's. From this point of view, a cross-point fault may have one of the following effects, depending on its location (AND or OR plane) and on its type (missing or additional) :

1. growth : a missing device in the AND array causes a literal to disappear from an implicant and hence the implicant covers more minterms.

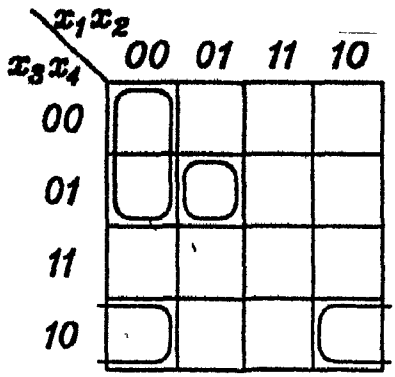
2. shrinkage : an additional device in the AND array causes a literal to appear in an implicant and therefore the number of minterms covered by the implicant is reduced.

3. disappearance : a missing device in the OR array causes an implicant to disappear from a function.

4. appearance : an additional device in the OR array causes a unwanted implicant to appear in some function.

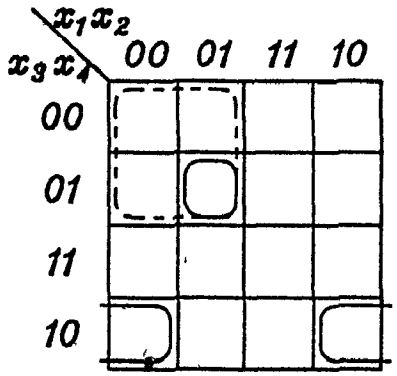
For example, consider one of the functions implemented by the PLA in figure 1.1, say g_1, and its K-map representation in the fault-free case, shown in figure 2.2a. A missing transistor at the intersection of bit line x_2 and product line p_4 causes the product to grow, as shown in figure 2.2b. Figures 2.2c, 2.2d, and 2.2e show examples of shrinkage, disappearance, and appearance faults, respectively.

One of the earliest algorithms introduced for PLA testing is the one presented in [6]. The algorithm selects a fault from the fault dictionary, generates a test vector for the fault, if it is testable, and then determines which other faults are detected by the newly generated test vector. It is pointed out in [6] that the order in which the faults are considered may have a considerable effect on the size of the test set. It has been noted that the closer a fault is to the primary outputs, the more tests it has. Therefore, the faults with fewer tests are considered first, that is, growth and shrinkage faults, in the hope that test vectors for these faults will accidentally detect a large number of the appearance and disappearance faults which are easier to detect.



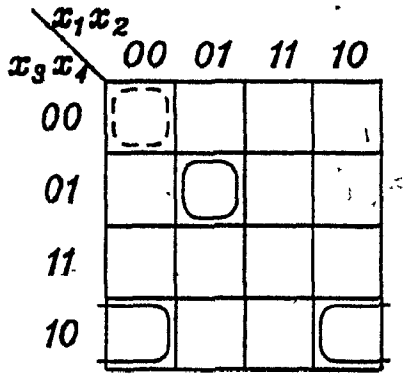
$g_1 = \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}\overline{x_2}\overline{x_3} + \overline{x_2}x_3\overline{x_4}$

a)



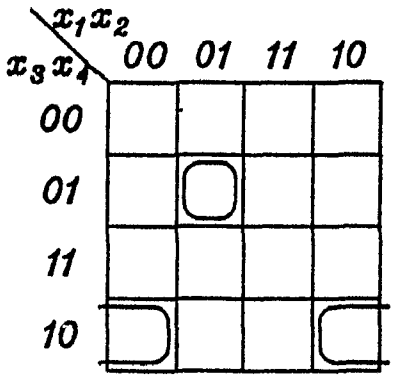
$g_1 = \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}\square\overline{x_3} + \overline{x_2}x_3\overline{x_4}$

b)



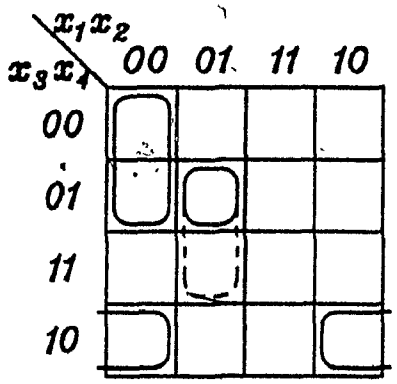
$g_1 = \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}\overline{x_2}\overline{x_3}\boxed{\overline{x_4}} + \overline{x_2}x_3\overline{x_4}$

c)



$g_1 = \overline{x_1}x_2\overline{x_3}x_4 + \square + \overline{x_2}x_3\overline{x_4}$

d)



$g_1 = \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}\overline{x_2}\overline{x_3} + \overline{x_2}x_3\overline{x_4}$
$\boxed{+\ \overline{x_1}x_2\overline{x_4}}$

e)

Figure 2.2 Functional Effects of CP Faults – Examples

A growth fault is caused by the disappearance of a device at some cross-point (i.e. of a literal from a product). In order to exercise a growth fault, the test pattern must activate the minterm(s) which have been added to the fault-free product as a result of the fault. These minterms are algorithmically determined by the "sharp product" [6] of the faulty and fault-free implicant. The operation returns the minterms which are covered by the faulty product but are not covered by the fault-free product. For the example in figure 2b, the fault-free product $\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$ has grown to the faulty $\overline{x_1} \cdot \overline{x_3}$ due to a missing transistor. The sharp operation on the corresponding cubes is :

$$0x0x \ \# \ 000x \ = \ 010x$$

Note that the cube resulting from this sharp operation contains the complement of the literal which disappeared from the fault-free product, while the other literals are the same as in the fault-free product. All the literals of a product represent bit lines which are fed into the same AND gate. Since a test vector for the growth fault has to activate some minterm in the cube returned by the sharp operation, the gate input corresponding to the missing literal ($\overline{x_2}$ in the example) has to be "0" while all the other inputs have to be "1" (AND-OR implementation). The important point is that the bit line corresponding to the tested cross-point carries a dominant value while the other bit lines participating in the product carry nondominant values.

A test vector, if it exists, is given by a sharp operation between the cube representing the minterm which exercise the fault, and all the other implicants of a function which uses the product line containing the tested cross-point. This is the sensitization part, and all the output lines which use the product containing the growth fault are considered sequentially until a test vector is found, or the fault is declared undetectable. For the example considered before and the output line g_1, a test vector is computed as follows :

$$((010x) \ \# \ 0101) \ \# \ x010 \ = \ 0100$$

This operation returns the minterms which are added to the function in the faulty case, and are not covered by any other product which participates in the function. Another test vector may be calculated in a similar way via the output line g_3. If the result is void

for all the output lines, then the fault is untestable. Similar calculations are required for the shrinkage faults.

The tests for appearance and disappearance faults are slightly easier to compute. The gates in the OR array are assumed to be directly observable and therefore only the fault exercising step is required. A test vector for a disappearance fault has to activate some minterm which is uniquely brought into the function by the tested product. A cube of such minterms can be calculated by performing a sharp operation between the tested product and all the other implicants of the function. Similarly, a test vector for an appearance fault has to activate some minterm which is covered by the tested product but not covered by any implicant of the function.

The same basic ideas for PLA test pattern generation and simulation can be found in [7]. The algorithm presented in [7] is designed for PLA's with two-input decoders. The values of the test vectors are computed at the level of bit lines rather than primary input lines, but, as pointed out in [7], the conversion to primary input values is a straight forward process. These features make the algorithm quite general : it can be used for PLA's implemented both with single input and double input decoders without any modification.

One of the more recent test generation algorithms for PLA's is PLATYPUS, presented in [9]. It combines biased random test pattern generation and deterministic test generation in an attempt to achieve the best balance between run time and test set minimality. It introduces heuristics for test compaction and uses the heuristic for cube ordering and the fault simulation algorithm introduced in [7]. The deterministic test pattern generation algorithm uses the sharp product as well, but the computational efficiency is improved (by using the Shannon's expansion theorem combined with the properties of unate functions). The algorithm and the heuristics are supported by extensive experiments performed on a set of 56 benchmarking PLA's.

The difficulty of test generation for PLA's is mainly due to very large gate fan-in and a relatively large number of redundant cross-points, which can be found in almost all practical PLA's. The testing problem can be simplified significantly by

designing the PLA to be easily testable. [14],[29],[16]. This is done by adding extra hardware, which increases the controllability and observability of all PLA cross-points, resulting in a PLA which requires less computational effort for test pattern generation. In the case of the design proposed in [14], the augmented PLA can be tested with universal test sets, and the coverage obtained is 100% of all single stuck-at and cross-point faults and most of the multiple stuck-at and cross-point faults. However, the size of the test set may become very large compared to the size of the test set for the non-easily testable PLA's. It is reported in [30] that for the 56 benchmarking PLA's [9], the easily testable PLA designs proposed in [31],[32],[29] require an average test set of 5876,3077 and 488 vectors, respectively, compared to an average of 436 vectors given by PLATYPUS [9]. Another aspect related to the easily testable PLA designs is signal degradation. From this point of view, it is desirable that in normal mode of operation the PLA does not have additional devices connected in series on inputs. For example, in the design reported in [29], each inverted bit line passes through a pass transistor which in normal mode of operation adds to the input resistance of the inverted bit lines. Thus, the easily testable PLA designs represent tradeoffs over fault coverage, silicon area overhead, size of the test set, and performance degradation.

2.4 Embedded PLA Testing – Classical Solutions

The algorithms discussed so far are all based on the cross-point fault model and are designed under the assumption that the PLA inputs are fully controllable and the PLA outputs are fully observable. This assumption does not hold for embedded PLA's. Therefore, to test embedded PLA's with test vectors generated on the specific fault model, extra hardware has to be provided to control/observe the PLA inputs/outputs, as proposed in [21] and [22] for the general case. There are various disadvantages related to such schemes. They require extra design effort, occupy extra silicon area, require extra pins, require long test pattern application time, and may result in performance degradation.

One alternative is to use the stuck-at fault model which allows deterministic test pattern generation for the entire circuit, without requiring additional controllabil-

ity/observability schemes. Such an approach has been proposed in [33]. The cross-point fault model has been discarded and the algorithm simply considers the set of collapsed stuck-at faults in the logic representation of the PLA. It is pointed out in [33] that the missing cross-point faults (growth and disappearance) can be mapped into stuck-at faults in the two-level gate model, and that the additional cross-point fault (shrinkage and appearance) is not significant in the MOS technology. The second statement can be supported by the results presented in [12], [34], and [35], for the single metal layer NMOS technology. Statistical analysis of defects, based on information from industry shows that the probability of a missing transistor fault or an additional transistor fault is three orders of magnitude smaller than the probability of a single line stuck-at zero, and two orders of magnitude smaller than the probability of bridging faults between product lines [12].

The fault set has been reduced by fault collapsing to three types of faults : stuck-at one faults on the inputs of the gates in the AND plane, stuck-at zero faults on the inputs of the gates in the OR plane, and stuck-at one faults on the PLA outputs. Various heuristics are used to improve the algorithm efficiency. For example, the test set is dynamically compacted and the faults are considered in a certain order, starting with the AND plane and ending with the PLA output stuck-at one faults. The algorithm and the heuristics are tailored for two-level functions and the experimental results [33] show that the algorithm has a good time and test set size performance. However, the initial assumptions limits its application to the MOS technology. The bridging faults are not considered at all, even though their probability of occurrence is the same order of magnitude as that of stuck-at faults. In this respect, it is mentioned in [33] that the problem of bridging faults will make the subject of future work.

Another approach, based on the stuck-at fault model, is presented in [11]. The cross-point faults are modeled as stuck-at faults by adding a two-input extra gate and an extra input which carries a constant value for each cross-point. This model has been used in [13] to study multiple faults in PLA's. For example, consider the PLA in

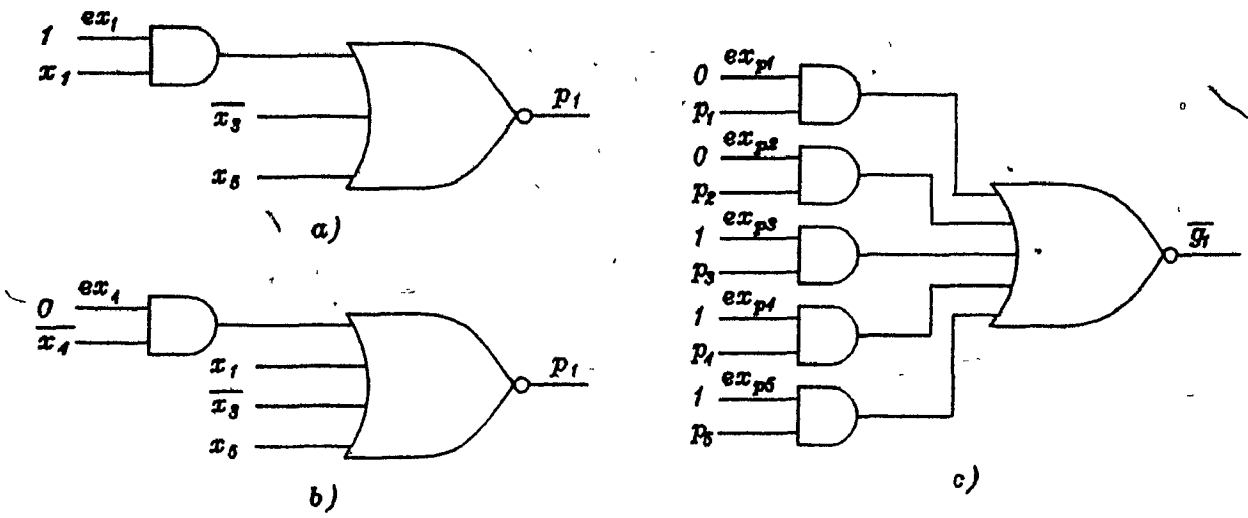


Figure 2.3 Stuck-at Models for Cross-Point Faults

figure 1.1 a) and its NOR-NOR implementation in figure 1.1 d) *. The missing cross-point fault at the intersection of $\overline{x_1}$ and product line p_1 can be modeled as shown in figure 2.3 *a*) by adding one extra two-input AND gate and one constant input ex_1. The missing cross-point fault is covered by the test vector for the fault ex_1 stuck-at zero [13],[11]. The stuck-at model for the additional cross-point fault at the intersection of product line p_1 and bit line $x4$ is shown in figure 2.3 *b*). The additional cross-point fault is covered by the stuck-at one fault on the extra line ex_4 feeding the two-input AND gate. The cross-point faults in the OR plane can be modeled in a similar way, as shown in figure 2.3 *c*). Since all the PLA cross-point faults can be modeled in this way, a complete single stuck-at test set generated for this model would cover all detectable CP faults. However, the size of the model is prohibitive. The PLA two-level logic model becomes a four-level model and a two-input gate is introduced for each crosspoint.

The bridging faults of interest [36] are modeled as stuck-at faults as well, by adding four two-input gates for each bridging fault. It is assumed that the bridge has a definite logic value, either AND or OR. For example, consider a bridging fault between two product lines p_1 and p_2. Assuming that the bridge has an AND logic effect, it can

* Recall that the NOR-NOR implementation requires inversion of inputs and outputs, with respect to the representation in figure 1.1.

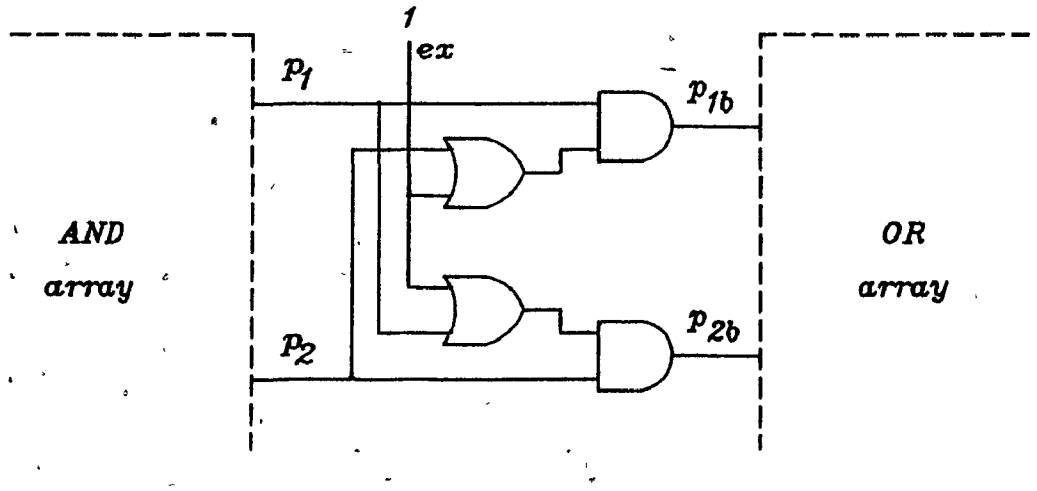


Figure 2.4 Stuck-at Model for Bridging Fault, AND Effect

be modeled as shown in figure 2.4. The bridge is covered by the test vector for the stuck-at zero fault on line *ex*. The bridging faults between bit lines or between output lines are modeled in the same way. Note that the model for each bridging fault requires four two-input gates and one constant input.

This method has been applied to three example PLA's and the results are reported in [11]. The gate counts for the models used are shown in table 2.1. Columns 2,3, and 4 correspond to the three example PLA's. Rows 1,2, and 3 show the number of input, product, and output lines; row 4 shows the number of gates in the two-level gate model of the PLA's; rows 5,6, and 7 show the gate count of the extended model for stuck-at plus cross-point faults, stuck-at plus bridging faults, and stuck-at plus cross-point plus bridging faults, respectively. Notice that only the bridging faults of interest have been considered in [11], and for example 3, the gate count in the extended model for stuck-at, cross-point, and bridging faults of interest is 28 times bigger than the two-level logic model.

The advantage of the method is that it allows test generation for PLA stuck-at, bridging and cross-point faults using the classical stuck-at tools, since a unique fault model is used. It also allows deterministic testing of embedded PLA's without requiring additional controllability/observability schemes. The main disadvantage is that the method requires a very large model of the PLA. The computational burden

	1	2	3	4
		Example 1	Example 2	Example 3
1	ip	22	17	16
2	pr	47	184	70
3	op	15	24	20
4	2-level model (nr. gates)	121	274	142
5	SA+CP model (nr. gates)	3874	n.a.	3478
6	SA+BR model (nr. gates)	n.a.	2380	691
7	SA+CP+BR model (nr. gates)	n.a.	n.a.	4027
n.a. : not available				

Table 2.1 Extended Model – Gate Counts

for a large model is even more significant when the PLA is treated as part of the larger circuit [22].

With respect to the idea of testing only for the stuck-at faults in the two-level model of the PLA, the following experiment is reported in [11]. A complete single stuck-at test set (T_{SA}) was generated on the two-level gate model of the PLA and then applied to the larger circuit which models cross-points and bridging faults as stuck-at faults. Notice however that this experiment is limited to just three PLA's and therefore it does not provide much generality. For example 3, the results show that T_{SA} covers 82% of cross-point faults and stuck-at faults (6% iredundant faults not covered) and 99.5% of the bridging faults of interest **and** stuck-at faults. For applications where a high coverage is required, this is a significant difference.

The third alternative proposed in the literature for embedded PLA testing is

Built-In Self Test (BIST). For general circuitry, BIST schemes consist basically of pseudo random test pattern generation and output response compression. The two parts are usually implemented using Linear Feedback Shift Registers and Multiple Input Shift Registers, respectively. The very large fan-in of the gates in PLA's makes these circuits random pattern resistant and the general BIST scheme has limited performance, unless exhaustive testing is performed [1]. As a result, PLA-specific BIST schemes have been proposed, taking advantage of the regular structure of the PLA's. In test mode, these designs change the structure of the PLA either by partitioning it or by adding extra circuitry to make the PLA easily testable.

The approach presented in [18] partitions the PLA into the AND and OR arrays and tests them separately. Test vectors are generated using non-linear feedback shift registers, which are augmented such that they can perform output response compression as well, thus implementing the so called built-in logic bloc observers (BILBO). The scheme is easy to implement but require high area overhead.

Various other BIST schemes are combined with easily testable PLA designs [17],[19],[37]. This allows the use of simple circuitry for test pattern generation and output response compression. In the limit, the PLA can be augmented such that the test vectors and/or the output response compression are independent of the functionality of the PLA. A representative approach in this respect is presented in [19]. The following circuitry is added to make the PLA easily testable :

- a shift register which controls the product lines. It is used in test mode to enable one single product line while the other product lines are set to propagating values.
- two control lines between the input decoders and product lines. In test mode, they are used to enable the inverted or direct bit lines.
- one or two product lines in the AND and OR arrays such that the number of devices and no devices on each bit line can be forced to be odd.
- one output line such that the number of devices on each product line in the

OR plane can be forced to be odd.

This additional circuitry allows PLA testing with the universal test set described in [19]. The output response is compressed into the cumulative parity bit sequence, which is a sequence of alternating 0's and 1's for the fault-free PLA. Therefore, the compressed output sequence can be generated by a toggle flip-flop, without requiring any storage, or it can be further compressed into two bits which give the count of 1-to-1 transitions and 0-to-0 transitions. The scheme is proved to cover all single cross-point, stuck-at and bridging faults. It is also proved that fewer than $2^{-(m+2n)}$ of the multiple faults may remain undetected, where m is the number of inputs and n is the number of outputs. It has been assumed that the effect of a bridging fault is AND.

The BIST scheme has been implemented in NMOS technology and the area overhead has been evaluated for some example PLA's. The area overhed is claimed to be the smallest [19] compared to other BIST schemes. Still, for medium size PLA's (20 inputs, 50 product lines, and 20 outputs) the area overhead is above 50%. For large PLA's (60 inputs, 250 product lines, and 50 outputs) the area overhead is 17%.

Clearly, the classical solutions to the problem of embedded PLA testing represent tradeoffs over various design and manufacturing aspects, such as design effort, test computation effort, silicon area, yield, extra pins, and testing time. The next chapter will introduce a new fault model which allows PLA testing using classical ATPG and simulation tools after minor changes, and allows testing of embedded PLA's without requiring BIST, additional controllability/observability schemes, or an enhanced model of the PLA.

Chapter 3 Conditional Stuck-At Fault Model

The conditional stuck-at (CSA) fault model introduced in this thesis satisfies the requirements of treating both general circuits and PLA's under the same fault model. Stuck-at faults are a subset of CSA faults. Moreover, the ATPG tools developed for the stuck-at fault model can be used for the CSA fault model after minor changes. For PLA's, cross-point and bridging faults are mapped into CSA faults using only the two-level gate model of the PLA.

Definition : A fault $(l_i/\alpha, l_j = \beta)$, where l_i and l_j are two lines in a circuit and $\alpha, \beta \in \{0,1\}$ is a conditional stuck-at (CSA) fault if l_i/α refers to the fault l_i stuck-at α and $l_j = \beta$ refers to the requirement that some test vector for the stuck fault l_i/α produces the value β on line l_j. This test vector is then said to detect the CSA fault $(l_i/\alpha, l_j = \beta)$.

The definition includes the null condition possibility corresponding to a normal stuck fault, where $(l_i/\alpha, l_j = \beta)$ is simply (l_i/α) and no l_j or β is specified. This type of CSA faults are going to be called null condition CSA faults. The expression "completely specified CSA fault" will be used whenever it is necessary to emphasize the fact that both the condition line and the condition value have to be specified, as opposed to the null condition CSA faults.

A test vector for the CSA fault $(l_i/\alpha, l_j = \beta)$ does not have to propagate the value on the condition line to some primary output. In the fault-free case, the condition may not have any effect on any primary output. In the faulty case, due to

non stuck-at defect, the value on the condition line l_j changes the value on the line l_i which is observable at some primary output, under the corresponding test pattern. It follows that the values α and β have to be determined such that, under the assumed non stuck-at fault, the interaction between lines l_i and l_j has the effect described above.

In general, a fault is detected by a test vector if some primary output PO_z is different from the fault-free primary output PO_z^{ff}. Under the single fault assumption, say (l_k/λ_k), this requirement reduces to observing, via some primary output, that the value λ_k on line l_k is different from the fault-free value λ_k^{ff}. Similarly, for the CSA faults, a fault which may not be a stuck-at fault is detected if the value λ_i on line l_i is observed to be different from the fault-free value λ_i^{ff} on l_i. Hence, for proofs and discussions related to the CSA faults $(l_i/\alpha, l_j = \beta)$, it is sufficient to focus on the two lines l_i and l_j since the part l_i/α ensures a propagation path from line l_i to some primary output and produces the value $\overline{\alpha}$ on line l_i.

Given a conditional stuck-at fault $(l_i/\alpha, l_j = \beta)$, it is possible that line l_j has to satisfy some requirement for sensitizing the path from l_i to some primary output. For example, if both l_i and l_j in the CSA fault $(l_i/0, l_j = 0)$ are input to some NOR gate, the condition has to be satisfied anyway for sensitizing the path for $l_i/0$. Hence, the condition $l_j = 0$ need not be specified in such a case since it is implicitly required by $l_i/0$. On the other hand, if the CSA fault is $(l_i/0, l_j = 1)$ and l_i, l_j are inputs to some NOR gate, there is a conflict between the sensitization requirements for $l_i/0$ and the condition $l_j = 1$. Thus, the CSA fault $(l_i/0, l_j = 1)$ specified above is untestable.

The relation between CSA faults and stuck-at faults is quite clear. For PLA's, it has to be shown that cross-point and bridging faults can be modeled as CSA faults.

3.1 Cross-Point Faults Modeled as CSA Faults

Some terms have to be defined before stating lemmas 1 and 2. A **gate line** in a PLA is the line which represents the output of some gate (either a product line or an output line). A **variable line** v with respect to some gate line g is a line on the next

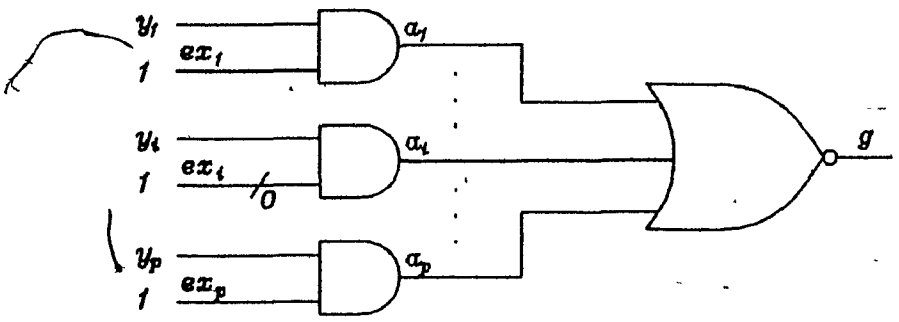


Figure 3.1 Missing CP Fault Modeling

lower level with respect to line g. For example, bit lines are variable lines with respect to product lines. Note that the value of a gate line g may not depend on a variable line v.

It has been mentioned in [33] that missing cross-point faults can be mapped into stuck faults (that is, CSA faults with null condition). This result is formalized below.

Lemma 1 : Any missing cross-point fault at the intersection of some variable line v and some gate line g is covered by the stuck-at fault v/α on the variable line v, where α is the nondominant value for the gate g.

Proof : Without losing any generality the lemma is going to be proven for the NOR-NOR implementation. Missing cross-point faults can be modeled as shown in figure 3.1 by adding one two-input AND gate and an extra input line ex for each used cross-point. The fault is modeled by a stuck-at zero fault on line ex_i. It is clear that $(ex_i/0)$ is equivalent to $(y_i/0)$ which means that a test vector for $(y_i/0)$ will also detect $(ex_i/0)$. Thus, the missing cross-point fault corresponding to the input y_i is detected by the test vector for the stuck fault $(y_i/0)$, which produces a dominant value on the tested input while the other inputs have nondominant values. The extra gates can be discarded since the fault $(y_i/0)$ is going to be tested for by T_{SA} anyway. □

The used PLA cross-points are represented in the two-level gate model as gate inputs. If, due to some defect, such an input does not exist, then a complete single

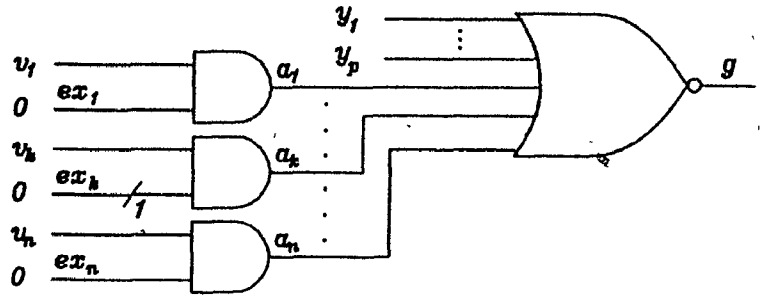


Figure 3.2 Additional CP Fault Modeling - NOR-NOR Implementation

stuck-at test set T_{SA} detects the fault. The unused PLA cross-points do not have any representation in the two-level gate model. In some sense, they represent inputs which do not exist. It is one of the applications of CSA fault model to allow testing for the non existence of such inputs. This can be achieved by producing a dominant value on the input line as a condition and observing the gate whose other inputs are at nondominant values. This is formally presented in lemma 2.

Lemma 2 : The additional cross-point fault at the intersection of some gate line g and any associated variable line v is detected by the test vector(s) for the CSA fault $(g/\alpha, v = \beta)$, where α and β take the values $\alpha = 0$, $\beta = 1$ for the NOR-NOR implementation.

Proof : As discussed before, it is sufficient to consider isolated gates since the part g/α in the CSA fault insures fault activation and propagation to some primary output. Consider the stuck-at model for additional cross-point faults of some gate line, as shown in figure 3.2. The stuck-at one fault on the extra line ex_k $(ex_k/1)$ will cover $(a_k/1)$ which is equivalent to $(g/0)$. The fault $(a_k/1)$ is covered by $(v_k/1)$ as well. Thus, a test vector for $(g/0)$ detects $(a_k/1)$ and either $(ex_k/1)$ or $(v_k/1)$. Since the objective is to test for $(ex_k/1)$, the test vector for $(g/0)$ has to produce a one on line (v_k). Hence the CSA fault $(g/0, v_k = 1)$. Thus, if the unwanted transistor exists, it is going to dominate the gate value and the effect will be observed at some primary output. □

Once it is decided that the CSA fault $(g/0, v = 1)$ models the corresponding additional cross-point fault, the model in figure 3.2 does not have to be considered any more since it applies to any unused cross-point. Thus, for additional cross-point faults,

$(g/0, v = 1)$ becomes a rule which is sufficient to generate the CSA faults from the personality matrix, and thus test these faults.

3.2 Bridging Faults Modeled as CSA Faults

There is increasing evidence that bridging faults represent a relatively large part of the defects which may occur in MOS IC's [34], [12],[35]. Bridging faults can be divided into two classes : bridges between wires on the same layer and bridges between wires on different layers. For PLA's, these two classes can be further divided as follows. On the same layer, bridging faults may occur between adjacent bit lines, between adjacent product lines or between adjacent output lines. On different layers there may be bridges between bit lines and product lines, and/or between product lines and output lines.

Statistical information from fabrication processes show that bridges may occur between more than two adjacent wires [12],[35]. However, for the purpose of this analysis, it has been assumed that the bridge occurs only between two wires. The analysis can be easily extended for the case of more than two wires shorted. The traditional assumption that the bridge has a well defined logic effect, either AND or OR [7],[11], has been adopted. It has also been assumed that the relative position in the layout of an input line with respect to the other input lines, and of a product line with respect to the other product lines, follows the position in the personality matrix. This assumption is necessary because only adjacent lines are considered as potential bridging faults.

It is possible that in the actual PLA layout, the distance between two adjacent parallel wires is so big that a short is very unlikely to occur, as opposed to some other wires which are very close, on the same layer. In this analysis, it has been assumed that a bridging fault may occur between any pair of adjacent bit lines, or product lines, or output lines.

In order to exercise a bridge it is necessary to produce complementary logic values on the two lines and to propagate the effect to some primary output. In general,

given the lines a and b in a circuit, the bridge between the two can be modeled either as $(a/\alpha, b = \alpha)$ or $(b/\alpha, a = \alpha)$. It is important to note here that the condition value is always the dominant value under the logic effect assumed. The associated stuck-at fault is always a stuck-at the dominant value such that the test vector creates a nondominant value on the stuck line. This guarantees that the short will change the value on the stuck-at line and not on the condition line, which makes the fault observable at some primary output via the sensitized path.

It has been found out however that, depending on the personality matrix, many of the bridging faults in PLA's are always detected by the test vectors for stuck-at faults in the two-level gate model. Consider the conditional stuck-at fault $(a/\alpha, b = \alpha)$ which models the short between lines a and b. If some sensitization requirement for the stuck-at fault a/α satisfies the condition $b = \alpha$ then then the CSA fault is detected by the test vector for the stuck-at fault a/α. Therefore, a better efficiency can be achieved if such cases are identified in advance.

Similarly, the condition in the CSA fault modeling some bridge may contradict some sensitization requirement for the stuck-at line. For example, assume that the two lines a and b are inputs of the same NOR gate and the effect of the short is OR. The CSA fault $(a/1, b = 1)$, modeling the bridging fault between a and b, is untestable because $a/1$ requires $b = 0$ and the CSA fault requires $b = 1$. In the test generation process, the significant amount of computation required by such untestable faults is wasted because no increase in coverage is achieved. Thus, each untestable fault identified in advance implies time savings in the test generation part, provided that the "identifying" operation is simple.

The following chapter presents a detailed analysis of the five types of bridging faults along with various patterns which may occur in personality matrices. The analysis is aimed at identifying the cases where the condition in a CSA fault which models some bridging fault is guaranteed to be satisfied by the sensitization requirements for the stuck-at fault, and some of the cases of untestable CSA faults.

Chapter 4 Single Bridging Faults

A PLA personality matrix contains both functionality and some layout information. Therefore, the analysis of the PLA bridging faults is based on the patterns which occur in the personality matrix. The analysis determines the implications of these patterns upon modeling shorts as null condition CSA faults.

4.1 Bit Lines

In the case of adjacent bit lines, there are two possibilities : the shorted lines belong to the same variable or the shorted lines belong to different variables. If the shorted lines belong to the same variable say x_i, the short is equivalent to the fault (x_i/α). Depending on the effect of the short, α may be zero (AND effect) or one (OR effect). Consequently, this kind of short is modeled by null condition CSA faults.

Suppose that the shorted bit lines belong to different variables, say a and b. Figure 4.1 shows the possibilities of using the adjacent lines $\overline{a}$ and b in products (p_0, p_1, p_2, p_3).

The short is modeled according to each of these possibilities (p_0, p_1, p_2, p_3) and for AND or OR effect.

4.1.1 Case p_0

If neither $\overline{a}$ nor b is used on a given product line, then the short does not have any effect on the product line. If this applies to all product lines then the short

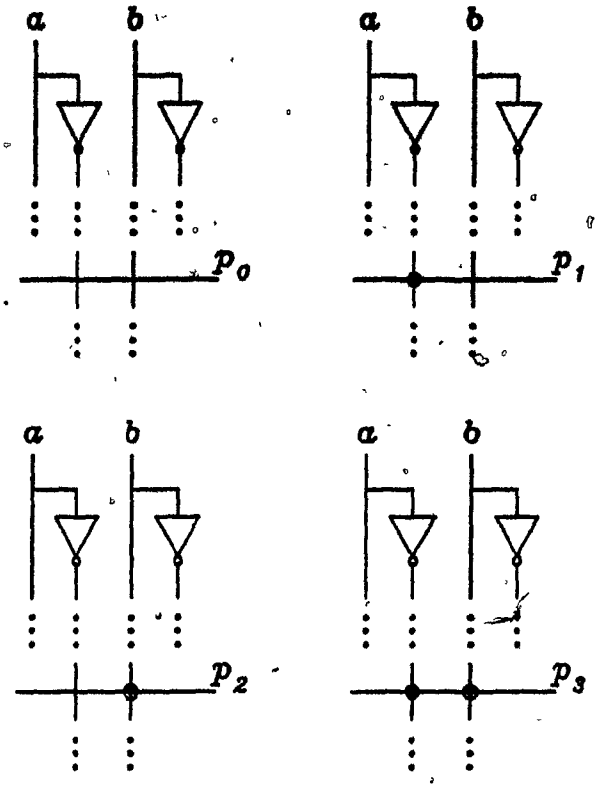


Figure 4.1 Adjacent Bit Lines Shorted

is an undetectable fault. This situation however will never occur in practice because if one or two bit lines are not used at all, then they simply do not exist.

4.1.2 Case p_1

Suppose that literal $\overline{a}$ is used in a product line but literal b is not used, as shown in figure 4.1 (p_1). Assuming a NOR-NOR implementation of the PLA, the fault-free p_1 (p_1^{ff}) is:

$$\begin{aligned} p_1^{ff} &= \overline{(i_1 + \ldots + \overline{a} + \ldots + i_k)} \\ &= \overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} \cdot a \end{aligned} \tag{4.1}$$

where $i_1, \ldots, i_k$ are the other inputs to that product line.

AND Effect

Assuming that the effect of the short is AND, the faulty p_1 (p_1^F) is :

$$\begin{aligned} p_1^F &= \overline{(i_1 + \ldots + \overline{a} \cdot b + \ldots + i_k)} \\ &= (\overline{i_1} \cdot \ldots \cdot \overline{i_k}) \cdot (a + \overline{b}) \\ &= (\overline{i_1} \cdot \ldots \cdot \overline{i_k}) \cdot a + (\overline{i_1} \cdot \ldots \cdot \overline{i_k}) \cdot \overline{b} \\ &= p_1^{ff} + (\overline{i_1} \cdot \ldots \cdot \overline{i_k}) \cdot \overline{b} \end{aligned} \tag{4.2}$$

The above expression leads to two subcases :

1. $\overline{b}$ is used in p_1
2. $\overline{b}$ is not used in p_1

If $\overline{b}$ is used in p_1 then p_1^F becomes:

$$\begin{aligned} p_1^F &= (\overline{i_1} \cdot \ldots \cdot b \cdot \ldots \cdot \overline{i_k}) \cdot a + (\overline{i_1} \cdot \ldots \cdot b \cdot \ldots \cdot \overline{i_k}) \cdot \overline{b} \\ &= (\overline{i_1} \cdot \ldots \cdot b \cdot \ldots \cdot \overline{i_k}) \cdot \\ &= p_1^{ff} \end{aligned} \tag{4.3}$$

Since p_1^{ff} is identical to p_1^F, the fault is undetectable through product line p_1.

If $\overline{b}$ is **not** used in p_1, then to detect the fault we must have $p_1^F \neq p_1^{ff}$ and the value on p_1 must be observable on at least one output line. It follows from the expression for p_1^F (4.2) that the the first requirement is satisfied if $(\overline{i_1} \cdot \ldots \cdot \overline{i_k}) \cdot \overline{b} = 1$ and $p_1^{ff} = 0$. Clearly, a test pattern which can detect the fault $\overline{a}/0$ and at the same time produce $b = 0$ would satisfy both requirements. In other words, this fault is modeled as $(\overline{a}/0, b = 0)$.

The same analysis can be done in terms of logic gates. Figure 4.2 shows the circuit model of the short. The effect of the short is as if the extra line ex were stuck-at

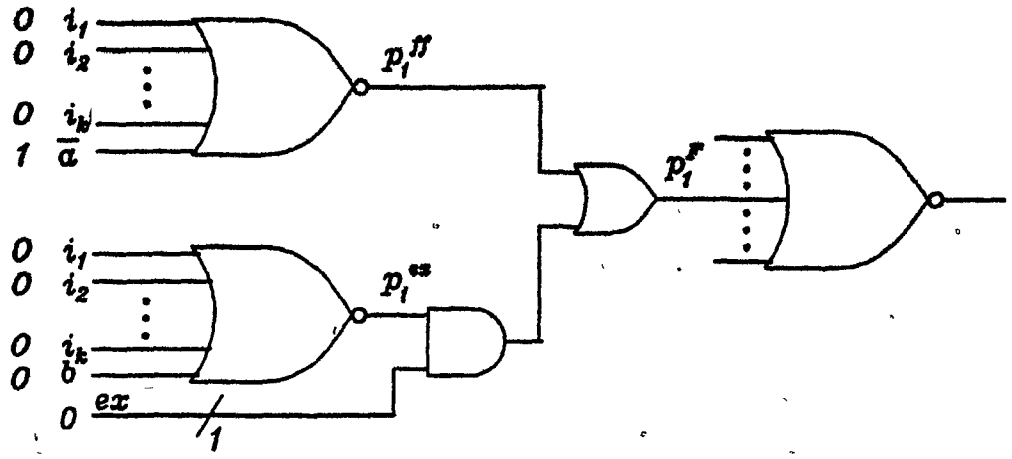


Figure 4.2 Adjacent Bit Lines; Case p_1; AND Effect

one. With the constant input zero for ex, in order to test for $ex/1$, the only requirement is a sensitized path from ex to some primary output. This implies $p_1^{ff} = 0$ and $p_1^{ex} = 1$, which translates into $\overline{a}/0$ and $b = 0$ which is in fact the CSA fault $(\overline{a}/0, b = 0)$. Note that a sensitization path for b instead of $\overline{a}$ cannot be used since in the fault-free case b is not used in product p_1.

OR Effect

Under the assumption that the effect of the short is OR, the faulty product p_1^F is :

$$\begin{aligned} p_1^F &= \overline{(i_1 + ... + (\overline{a} + b) + ... + i_k)} \\ &= (\overline{i_1} \cdot .... \cdot a \cdot .... \cdot \overline{i_k}) \cdot \overline{b} \\ &= p_1^{ff} \cdot \overline{b} \end{aligned} \tag{4.4}$$

1. If $\overline{b}$ is used on product line p_1, then p_1^F becomes :

$$\begin{aligned} p_1^F &= (\overline{i_1} \cdot .... \cdot a \cdot .... \cdot \overline{i_k}) \cdot b \cdot \overline{b} \\ &= 0 \end{aligned} \tag{4.5}$$

The short has the same effect as a stuck-at zero fault on line p_1 and it is modeled by a null condition CSA fault, meaning that any T_{SA} will detect this bridging fault.

2. If $\overline{b}$ is not used on product line p_1, then a similar analysis as the one for the AND effect can easily show that this fault is covered by the test vector for the CSA fault $(\overline{a}/1, b = 1)$.

4.1.3 Case p_2

The analysis for the case p_2 is very similar to to the analysis for the case p_1 presented above. In fact, one can be derived from the other by simply replacing $\overline{a}$ with b and vice versa. For this reason, the details of the analysis for the case p_2 are not presented. The results, however, are shown in table 4.1.

4.1.4 Case p_3

If combination p_3 occurs on two adjacent input variables, then the fault-free product p_3^{ff} is :

$$p_3^{ff} = \overline{(i_1 + i_2 + ... + \overline{a} + b + ... + i_k)} \tag{4.6}$$
$$= \overline{i_1} \cdot \overline{i_2} \cdot ... \cdot a \cdot \overline{b} \cdot \overline{i_k}$$

AND Effect

If the effect of the short between lines $\overline{a}$ and b is AND, then the faulty product p_3^F is :

$$p_3^F = \overline{(i_1 + i_2 + ... + \overline{a} \cdot b + ... + i_k)} \tag{4.7}$$
$$= \overline{i_1} \cdot \overline{i_2} \cdot ... \cdot (a + \overline{b}) \cdot \overline{i_k}$$
$$= (\overline{i_1} \cdot \overline{i_2} \cdot ... \cdot a \cdot \overline{i_k}) + (\overline{i_1} \cdot \overline{i_2} \cdot ... \cdot \overline{b} \cdot \overline{i_k})$$

In order to test for this fault, $i_1, ..., i_k$ have to be set to propagating values such that it can be observed if there is a sum (fault-free) or a product (faulty) between $\overline{a}$ and b (expressions 4.6 and 4.7, respectively). The fault can be exercised by producing

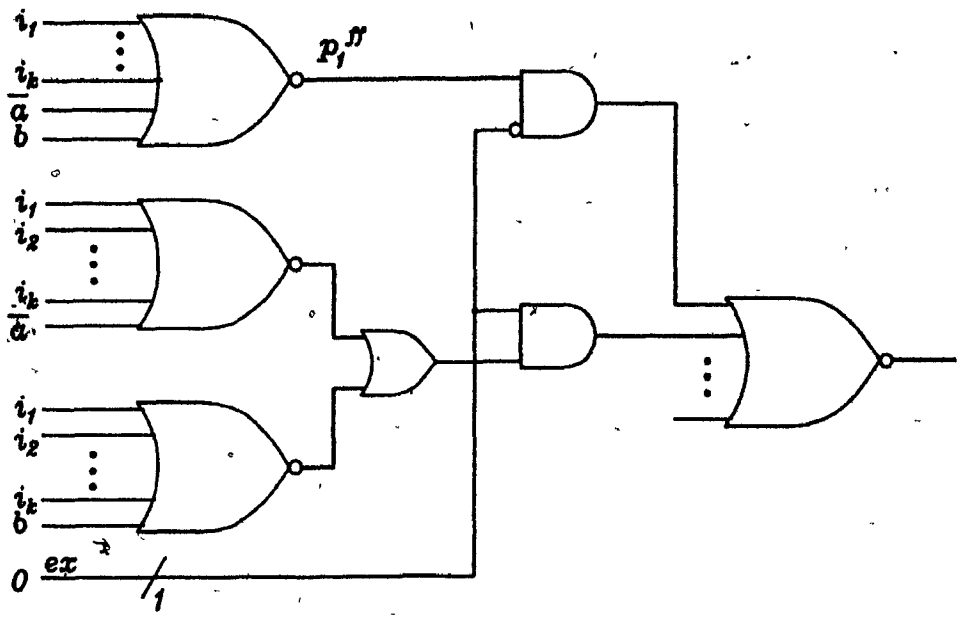


Figure 4.3 Adjacent Bit Lines ; Case p_3 ; AND Effect

complementary logic values on the two lines. Since both lines are inputs to the same NOR gate, the test vector for either $\overline{a}/0$ or $b/0$ will produce complementary logic values on $\overline{a}$ and b and obviously satisfy the sensitization requirements. Hence, the short is modeled as a null condition CSA fault.

In terms of logic gates, the short can be modeled as shown in figure 4.3. The short has the same effect as the stuck-at one fault on the extra line ex. It can be easily seen that $ex/1$ will be detected by the test vector(s) for $\overline{a}/0$, or $b/0$.

OR Effect

If the effect of the short is OR, then the faulty product p_3^F becomes :

$$\begin{aligned} p_3^F &= \overline{(i_1 + i_2 + ... + \overline{a} + b + ... + i_k)} \\ &= \overline{i_1} \cdot \overline{i_2} \cdot ... \cdot a \cdot \overline{b} \cdot \overline{i_k} \\ &= p_3^{ff} \end{aligned} \tag{4.8}$$

Thus, the fault is undetectable through product line p_3.

4.1.5 Combinations $p_1 - p_2 - p_3$

The personality patterns discussed individually so far may occur in practice in various combinations. It is thus necessary to analyze the implications of such combinations.

Suppose p_1 and p_2 occur on two adjacent input variables, say a and b. In this case, the effect of the fault may be propagated on two paths which may be completely or partially different. Thus, if the short has to be modeled as a completely specified CSA fault, there are two conditional stuck-at faults which might detect the short: $(\overline{a}/\alpha, b = \alpha)$ and $(b/\alpha, \overline{a} = \alpha)$, $\alpha \in \{0,1\}$. These two CSA faults correspond to the patterns p_1 and p_2 respectively. Certainly, the ATPG tool will consider the second CSA fault only if the first is undetectable.

If pattern p_3 appears in some combination with p_1 or p_2, then the case has to be analyzed for both AND and OR effects of the short. If the effect is AND, then the short is modeled as null condition CSA fault through the pattern p_3 (as shown for the case p_3). Thus, there is no need to test for some conditional stuck-at fault (via p_1 or p_2) which can also model the short.

If the effect is OR, the fault is undetectable through product line p_3 and the fault might be detected only via p_1 or p_2 which appear in the combination. The analysis for these cases has already been carried out and it shows that the short can either be modeled as a null condition CSA fault, or as a completely specified CSA fault.

4.2 Product Lines

The case of product lines involves fewer relevant personality patterns since no product line appears both complemented and uncomplemented, as in the case of input lines. Two adjacent product lines can be used in output functions as shown in figure 4.4. It has to be noted that g_1, g_2, and g_3 refer both to some personality pattern as shown in figure 4.4, and to PLA output lines before the final inversion (the implementation

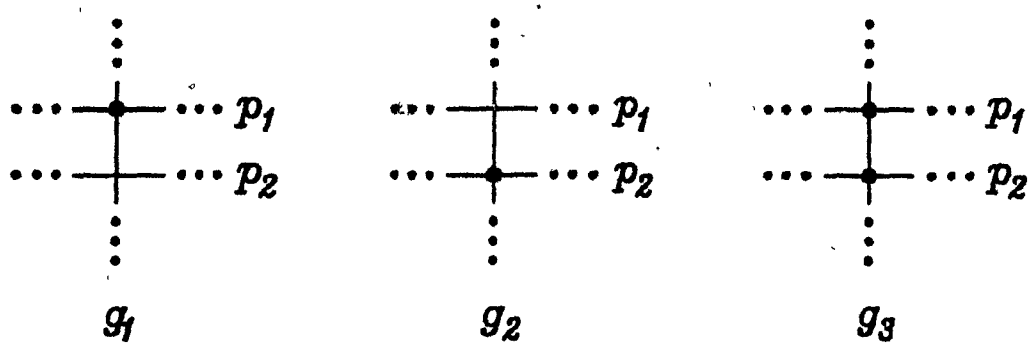


Figure 4.4 Adjacent Product Line Patterns

assumed is NOR-NOR). The analysis of the basic patterns g_1, g_2 and g_3, and and various patterns which may occur is quite similar to that for the patterns on bit lines.

4.2.1 Case g_1

If pattern g_1 occurs, then the fault-free output line g_1^{ff} (before the final inversion) is :

$$\begin{aligned} g_1^{ff} &= \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1} \\ &= \overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_1} \end{aligned} \tag{4.9}$$

Where π_i's are the other product terms on which the function g_1 depends.

AND Effect

Assuming an AND effect of the short, the faulty output line g_1^F is :

$$\begin{aligned} g_1^F &= \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1 \cdot p_2} \\ &= \overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot (\overline{p_1} + \overline{p_2}) \\ &= (\overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_1}) + (\overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_2}) \\ &= g_1^{ff} + (\overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_2}) \end{aligned} \tag{4.10}$$

In order to detect the short, it has to be observed at some primary output that $g_1^{ff} \neq g_1^F$. At the same time, a nondominant value under the assumed effect of

the short has to be produced on p_1. It follows from expression 4.10 that $g_1^F \neq g_1^{ff}$ if $\overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_2} = 1$ and $g_1^{ff} = 0$. Clearly, a test vector for the fault $p_1/0$ will produce an observable 1 on p_1. If this test vector also produces a 0 on p_2 then $\overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_2} = 1$ while $g_1^{ff} = 0$ and all the requirements are satisfied. In other words, the short is detected by a test vector for the CSA fault $(p_1/0, p_2 = 0)$.

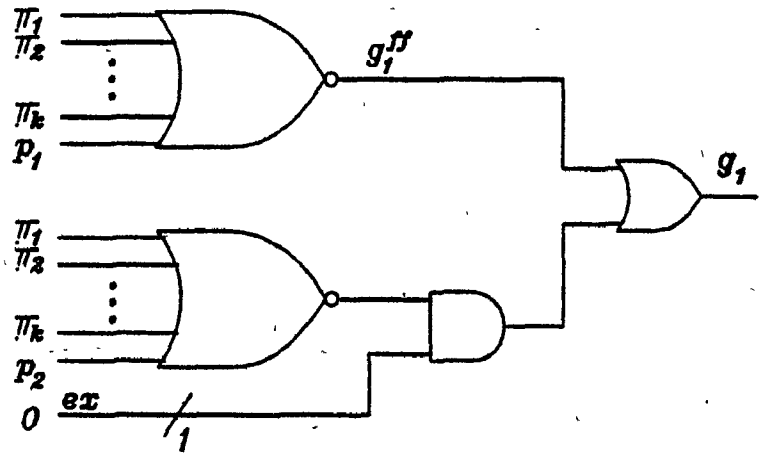


Figure 4.5 Adjacent Product Lines; Case g_1; AND Effect

The same analysis can be done on the gate-level model of the short, as shown in figure 4.5.

OR Effect

If the effect of the short is OR, then g_1^F is :

$$g_1^F = \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1 + p_2} \tag{4.11}$$

The short is detected if a nondominant value (zero for the OR effect) can be observed for the fault-free case on p_1 while p_2 has a dominant value (one for the OR effect). These requirements are expressed in the CSA fault $(p_1/1, p_2 = 1)$.

Case g_2 is very similar to case g_1 and results are shown in table 4.1 .

4.2.2 Case g_3

If pattern g_3 occurs on two adjacent product lines, the fault-free output line g_3 is :

$$g_3^{ff} = \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1 + p_2}$$
$$= \overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot \overline{p_1} \cdot \overline{p_2} \quad (4.12)$$

AND Effect

The faulty output line is :

$$g_3^F = \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1 \cdot p_2}$$
$$= \overline{\pi_1} \cdot \overline{\pi_2} \cdot ... \cdot \overline{\pi_k} \cdot (\overline{p_1} + \overline{p_2}) \quad (4.13)$$

It follows from expressions 4.12 and 4.13 that to detect this fault, it has to be observed whether there is a product $\overline{p_1} \cdot \overline{p_2}$ or a sum $\overline{p_1} + \overline{p_2}$ on g_3. If all π_i's, $(i = 1..k)$ have propagating values, the fault is detected if p_1 and p_2 have complementary logic values. Note that all these requirements are satisfied by the test vectors for the stuck-at faults $p_1/0$ and/or $p_2/0$. Therefore, no extra condition is required.

OR Effect

In this case, the faulty output line is :

$$g_3^F = \overline{\pi_1 + \pi_2 + ... + \pi_k + p_1 + p_2}$$
$$= g_3^{ff} \quad (4.14)$$

Hence, the short is undetectable through the output line g_3.

4.2.3 Combinations $g_1 - g_2 - g_3$

The basic patterns analyzed so far are very likely to occur in various combinations. As a consequence, there may exist a second alternative for testing the same short.

If g_1 and g_2 occur on the same adjacent product lines, then the fault can be tested either as a stuck-at fault on line p_1 with a condition for p_2 or as a stuck-at fault on line p_2 with a condition on line p_1. This is the only case when two CSA faults model the same product line short. Note that the number of occurrences of g_1 or g_2 is the number of paths on which the effect of the short can be propagated to some primary output.

If pattern g_3 is encountered along with g_1 and/or g_2, then if the effect of the short is AND, the fault is detected by any T_{SA}, as discussed for the case g_3. If the effect is OR, then the short has to be modeled by some CSA fault involving patterns g_1 or g_2, since it is undetectable through g_3.

4.3 Output Lines

The short between two adjacent output lines has to be modeled as a fault on the outputs of some gates on the second level. This assumes that there is no information about the circuitry being fed by the PLA outputs. Thus, there are always two completely specified CSA faults which model the same output line short. If the PLA is not embedded, the outputs are directly observable and only one CSA fault (either one) is sufficient for testing the short. In this case, the short is detected whenever the two output lines have different values in the fault-free case, independently on the logic effect of the short. If the PLA is embedded and the outputs are not directly observable, then both CSA faults have to be considered since either one may be untestable.

As discussed in section 3.2 for the general case, the CSA faults which model a short with **AND** effect are $(a/0, b = 0)$ and $(b/0, a = 0)$. For the case of two PLA

output lines g_i and g_j, the CSA faults are $(g_j/0, g_j = 0)$ and $(g_i/0, g_j = 0)$. For the case of **OR** effect, the CSA faults are $(g_j/1, g_j = 1)$ and $(g_i/1, g_j = 1)$.

4.4 Cross–Point Shorts

The name *cross-point shorts* (CPS) refers to shorts between wires on different layers at the crossover point of bit lines and product lines or product lines and output lines. Recent research work ([12], [35]) has shown that shorts between wires on different layers are less likely to occur. Even though the probability of such a short to happen is three orders of magnitude smaller than the probability of shorts on the same layer [35], some applications may require that these defects as well be covered.

The analysis assumes that the logic effect of the short is either AND or OR. As discussed in chapter 3, there are two CSA faults which can model the short. Consider for example a short between product line p_1 and bit line x_4 in the example PLA shown in figure 1.1. As discussed in section 3.2, there are two CSA faults which can model the same crosspoint short: $(p_3/\alpha, i_x = \alpha)$ or $(i_x/\alpha, p_3 = \alpha)$, where $\alpha = 0$ if the short is AND type and $\alpha = 1$ if the short is OR type. Similarly, a short between a product line p_j and an output line g_k can be modeled as $(p_j/\alpha, g_k = \alpha)$ or $(g_k/\alpha, p_j = \alpha)$

The layout information available from the personality matrix makes it possible to derive the CSA faults which model crosspoint shorts. It has been established so far that any bridging fault can be modeled as a completely specified CSA fault. In some cases, however, this may not be necessary since a null condition CSA fault is sufficient. The following two sections analyze the cross-point shorts under various personality patterns in order to determine the cases where a completely specified CSA fault is required and to derive the corresponding CSA fault.

4.4.1 Bit and Product Lines

Consider a cross-point short between product line p_i and bit line a. The fault-free product line p_i is :

$$p_i^{ff} = \overline{i_1 + i_2 + ... + i_k}$$

$$= \overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} \tag{4.15}$$

OR Effect

Under the assumption of OR effect, the faulty product p_i^F is :

$$\begin{aligned} p_i^F &= \overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} + a \\ &= p_i^{ff} + a \end{aligned} \tag{4.16}$$

Depending on the relationship between the input a and the inputs $i_1 \ldots i_k$, there are three possibilities :

1. none of i_j's is a (nor $\overline{a}$);
2. some i_j is a ;
3. some i_j is $\overline{a}$;

Case 1

There is no relation between a and $i_1 \ldots i_k$. It follows from expression 4.16 that in order to detect the fault we must have $p_i^{ff} = 0$, $a = 1$ and observe p_i at some primary output. Clearly, a test vector for the CSA fault $(p_i/1, a = 1)$ will satisfy these conditions and detect the short.

Case 2

The faulty product is :

$$\begin{aligned} p_i^F &= \overline{i_1 + a + i_2 + \ldots + i_k} + a \\ &= \overline{i_1} \cdot \overline{a} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} + a \\ &= \overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} + a \end{aligned} \tag{4.17}$$

A test vector for the fault $a/0$ will produce a zero on p_i in the fault-free case. In the presence of the short, it will produce a one on p_i and the short will be detected. Thus, a null condition CSA fault is sufficient to model the short.

Case 3

The faulty product is :

$$\begin{aligned} p_i^F &= \overline{i_1 + \overline{a} + i_2 + \ldots + i_k} + a \\ &= \overline{i_1} \cdot a \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} + a \\ &= a \end{aligned} \tag{4.18}$$

In order to detect the fault, we must have $\overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} = 0$, $a = 1$, and a sensitized path to some primary output. All these requirements are satisfied by a test vector for the fault $i_j/0$, $i_j \neq \overline{a}$. Thus, a null condition CSA fault can model the short.

AND Effect

The faulty product line p_i^F becomes :

$$\begin{aligned} p_i^F &= \overline{i_1 + i_2 + \ldots + i_k} \cdot a \\ &= \overline{i_1} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} \cdot a \end{aligned} \tag{4.19}$$

Again, there are three possibilities :

1. none of i_j's is a (nor $\overline{a}$);
2. some i_j is a ;
3. some i_j is $\overline{a}$;

Case 1

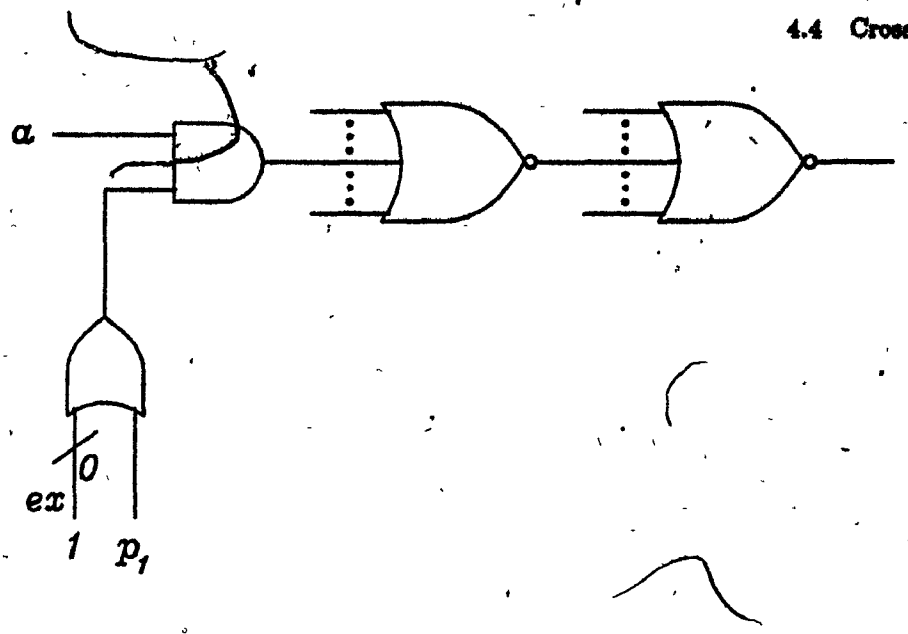


Figure 4.6 Bit and Product Line - Fault $(a/\alpha, p_i = \beta)$ (AND Effect)

The product p_i does not depend on a or $\overline{a}$. A similar analysis as in the case of OR effect leads to the conclusion that the short is going to be detected by the test vector for the fault $(p_i/0, a = 0)$.

Case 2

The faulty product is :

$$\begin{aligned} p_i^F &= \overline{i_1} \cdot \overline{a} \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} \cdot a \\ &= 0 \end{aligned} \tag{4.20}$$

Clearly, the short is covered by the null condition CSA fault $(p_i/0)$.

Case 3

Being a function of $\overline{a}$, the product p_2 becomes :

$$\begin{aligned} p_i^F &= \overline{i_1} \cdot a \cdot \overline{i_2} \cdot \ldots \cdot \overline{i_k} \cdot a \\ &= p_2^{ff} \end{aligned} \tag{4.21}$$

Thus, the fault $(p_i/\alpha, a = \alpha)$ is undetectable. However, it may be detectable via the fault $(a/\alpha, p_i = \alpha)$. The logic circuit image of this model is shown in figure 4.6.

The short is detected by the test vector for the CSA fault $(a/0, p_i = 0)$. This will produce a one on line a and a zero on $\overline{a}$. $a = 0$ is a nondominant value in the expression for p_i and thus there is no contradiction.

4.4.2 Product and Output Lines

The analysis is very similar to the case *Bit and Product Lines*. The only difference is that there is no inverted product line, the way bit lines are inverted in the AND plane.

The fault-free line g_i^{ff} is :

$$\begin{aligned} g_i^{ff} &= \overline{p_1 + p_2 + ... + p_k} \\ &= \overline{p_1} \cdot \overline{p_2} \cdot .... \cdot \overline{p_k} \end{aligned} \tag{4.22}$$

OR Effect

The faulty line g_i^F is :

$$\begin{aligned} g_i^F &= \overline{p_1} \cdot \overline{p_2} \cdot .... \cdot \overline{p_k} + \pi \\ &= g_i^{ff} + \pi \end{aligned} \tag{4.23}$$

Case 1

The output line g_i is independent on π. The short is going to be detected if $g_i^{ff} = 0$, $\pi = 1$ (expression 4.23), and if g_i is observable at some primary output. In other words, the short is modeled by the CSA fault $(g_i/1, \pi = 1)$.

Case 2

The output line g_i depends on π (used crosspoint). The faulty output line becomes :

$$\begin{aligned} g_i^F &= \overline{p_1 + \pi + p_2 + ... + p_k} + \pi \\ &= \overline{p_1} \cdot \overline{\pi} \cdot \overline{p_2} \cdot .... \cdot \overline{p_k} + \pi \\ &= \overline{p_1} \cdot \overline{p_2} \cdot .... \cdot \overline{p_k} + \pi \end{aligned} \tag{4.24}$$

As for the general case, the short can be modeled as $(g_i/1, \pi = 1)$. This CSA fault is detected by the test vector for the CSA fault $(\pi/0, \pi = 1)$, since $g_i/1$ dominates $\pi/0$. But, a test vector for $\pi/0$ produces $\pi = 1$ anyway, so no condition has to be specified for this case.

AND Effect

Assuming that the effect of the short is AND, the faulty output line g_i^F is :

$$\begin{aligned} g_i^F &= \overline{p_1 + p_2 + \ldots + p_k} \cdot \pi \\ &= \overline{p_1} \cdot \overline{p_2} \cdot \ldots \cdot \overline{p_k} \cdot \pi \end{aligned} \tag{4.25}$$

Case 1

None of p_i's is π. The analysis is similar to that for OR effect, case 1. The short will be detected by the test vector for the CSA fault $(g_i/0, \pi = 0)$.

Case 2

If $p_i = \pi$, then g_i^F becomes :

$$\begin{aligned} g_i^F &= \overline{p_1} \cdot \overline{p_2} \cdot \overline{\pi} \ldots \cdot \overline{p_k} \cdot \pi \\ &= 0 \end{aligned} \tag{4.26}$$

Thus, the short is modeled by the null condition CSA fault $(g_i/0)$.

4.5 Conclusion

The PLA regularity and implicit layout information enables one to efficiently model bridging faults as CSA faults. The results of the PLA bridging fault analysis are summarized in tables 4.1 and 4.2. Clearly, this analysis does not identify all the untestable single bridging faults, but provides a simple method to screen a subset of the untestable bridges. Even though a NOR-NOR implementation has been assumed for the PLA, these results are valid for the AND-OR implementation as well.

1. Adjacent bit lines shorted	Bit lines belong to the same var.	OR	null cond. CSA		
		AND	null cond. CSA		
	Bit lines on different variables	bit lines not used	Undetectable		
		p1	AND	$\bar{b}$ used	undetectable
				$\bar{b}$ unused	$(\bar{a}/0, b=0)$
			OR	$\bar{b}$ used	null cond. CSA
				$\bar{b}$ unused	$(\bar{a}/1, b=1)$
		p2	AND	a used	undetectable
				a unused	$(b/0, \bar{a}=0)$
			OR	a used	null cond. CSA
				a unused	$(b/1, \bar{a}=1)$
		p3	AND	null cond. CSA	
			OR	undetectable	
		p1 & p2 & p3	indep. alternatives: p1, p2, p3		
2. Adjacent word lines shorted	g1		AND	$(p_1/0, p_2=0)$	
			OR	$(p_1/1, p_2=1)$	
	g3		AND	null cond. CSA	
			OR	undetectable	
	g1 & g2 & g3		indep. alternatives: g1,g2,g3		
3. Adj. output lines shorted	AND		$(g_i/0, g_j=0)$ or $(g_j/0, g_i=0)$		
	OR		$(g_i/1, g_j=1)$ or $(g_j/1, g_j=0)$		

Table 4.1 Bridging Faults Between Parallel Wires on the Same Layer

The tables represent in fact a simple set of rules for deriving the CSA faults from the personality matrix. Once this is done, only the two-level logic model of the circuit is used for test pattern generation, as it will be described in the section on experiments and results.

4. Crosspoint shorts in the AND plane	OR effect	1: "a" not used	$(p/1, a = 1)$
		2: "a" used	null cond. CSA
		3: "$\overline{a}$" used	null cond. CSA
	AND effect	1: "a" not used	$(p/0, a = 0)$
		2: "a" used	null cond. CSA
		3: "$\overline{a}$" used	undetectable
5. Crosspoint shorts in the OR plane	OR effect	1: "p" not used	$(g/1, \overline{p} = 1)$
		2: "p" used	null cond. CSA
	AND effect	1: "p" not used	$(g/0, p = 1)$
		2: "p" used	null cond. CSA

Table 4.2 Bridging Faults Between Lines on Different Layers

Chapter 5 Experiments and Results

The analysis presented in chapters three and four has been used to implement an algorithm to generate the CSA faults for PLA's. This algorithm has been applied to the 56 benchmarking PLA's [9]. Test vectors were generated for the CSA faults in a subset of these examples. In each case, the set of faults contains both CSA faults with null condition and completely specified CSA faults. The ATPG tool used to generate test vectors is an existing implementation of PODEM [24] which has been modified such that it processes CSA faults as well. Unfortunately, the ATPG tool available is quite inefficient and the very large PLA's in the set of 56 benchmarking examples could not have been processed in a reasonable amount of time. For comparison with a state of the art ATPG tool [38], table 5.1 shows the time performance of the PODEM implementation used in these experiments, versus SOCRATES [38] for some of the well known benchmarking circuits introduced in [39]. For these reasons, the modified PODEM implementation was run only on 45 small and medium size PLA's out of the set of 56 PLA's. The implementation of the algorithm to generate the CSA faults and the modifications for the ATPG tool are the contribution of the author. The programs have been written in "C" run on a SUN 3/50 system.

5.1 CSA Faults

The input for the algorithm which generates the completely specified CSA faults is the PLA personality matrix. The output specifies the stuck line, the stuck value, the condition line and the condition value for each CSA fault. The lines are specified

Circuit	SOCRATES total time	PODEM total time
c432	3.7	279.67
c499	8.1	241.66
c880	5.7	159.50
c1355	21.9	1209.80
c1908	33.1	1063.52
c2670	69.3	6601.68
c3540	62.0	7494.29

Table 5.1 Calibration

as x_1 to x_n, p_1 to p_m, and F_1 to F_p, for PLA input lines, product lines, and output lines, respectively. Therefore, in the case of embedded PLA's, The ATPG program has to accept these names for lines which are internal to the larger circuit.

In the data structure, each input cube is represented by a linked list of 32-bit word blocks. In each block, two words are used to encode the literals in the cube. Therefore, each block can represent up to 31 entries in the input cube. The 32^{nd} bit overlaps the 1^{st} bit in the next block and it is used for checking the personality patterns on two adjacent lines represented on two different blocks, for the case of bridging faults. The OR plane is represented in a similar way. In this case, each output column is mapped into a list of words. This representation of the PLA allows a very efficient use of the memory and fast bit-wise operations.

The missing cross-point faults are covered by CSA faults with null condition (i.e. normal stuck-at faults), as stated in lemma 1, and PODEM will inherently consider them. The CSA faults for the additional cross-point faults are generated by simply traversing the list and identifying the non-connections. Two counters have to be kept, one for the input/product line count for the AND plane, and the other for the product/output line count for the OR plane. The words are simply shifted and ANDed with a mask to determine whether there is a connection or not. The CSA faults for the cross-point shorts are generated in the same pass by observing the rules summarized in table 4.2. As discussed in chapter three, the effect of shorts may be propagated on

	IP	PR	OP	time (sec)
add6	12	355	7	4.62
adr4	8	75	5	0.62
alu1	12	19	8	0.42
alu2	10	68	8	1.02
alu3	10	66	8	0.98
apla	10	25	12	0.40
bc0	26	179	11	4.04
bca	26	180	46	6.62
bcb	26	156	39	5.00
bcc	26	137	45	4.86
bcd	26	117	38	4.64
chkn	29	140	7	4.00
co14	14	14	1	0.24
cps	24	162	109	15.72
dc1	4	9	7	0.22
dc2	8	39	7	0.54
dist	8	120	5	1.20
dk17	10	18	11	0.40
dk27	9/8	10	9	0.30
dk48	15	21	17	0.62
exep	30	109	62	8.04
f51m	8	76	8	1.06
gary	15	107	11	2.14
in0	15	107	11	2.34
in1	16	104	17	1.86
in2	19	135	10	3.16
in3	35	74	29	4.02
in4	32	212	20	9.56

	IP	PR	OP	time (sec)
in5	24	62	14	2.02
in6	33	54	23	2.76
in7	26	54	10	1.92
jbp	36	122	57	9.78
misg	56	69	23	4.68
mish	94	82	34	10.58
mlp4	8	127	8	1.38
opa	17	79	61	4.40
radd	8	75	5	0.82
rckl	32	32	7	1.16
rd53	5	31	3	0.34
rd73	7	127	3	0.96
risc	8	28	31	1.04
root	8	57	5	0.66
sqn	7	38	3	0.44
sqr6	6	50	11	1.00
ti	43	213	67	20.20
tial	14	579	8	9.06
vg2	25	110	8	3.06
wim	4	9	7	0.30
x1dn	27	110	6	2.98
x2dn	82	104	47	14.42
x6dn	38	81	5	3.82
x7dn	66	538	15	46.70
x9dn	27	120	7	3.42
z4	7	39	4	0.62
Z5xp1	7	65	9	0.98
Z9sym	9	84	1	0.90

Table 5.2 Time Required for CSA Generation

two paths. Therefore, a new field is used in the CSA fault specification, indicating this possibility. If the fault is untestable via one path, the ATPG tool will try the second

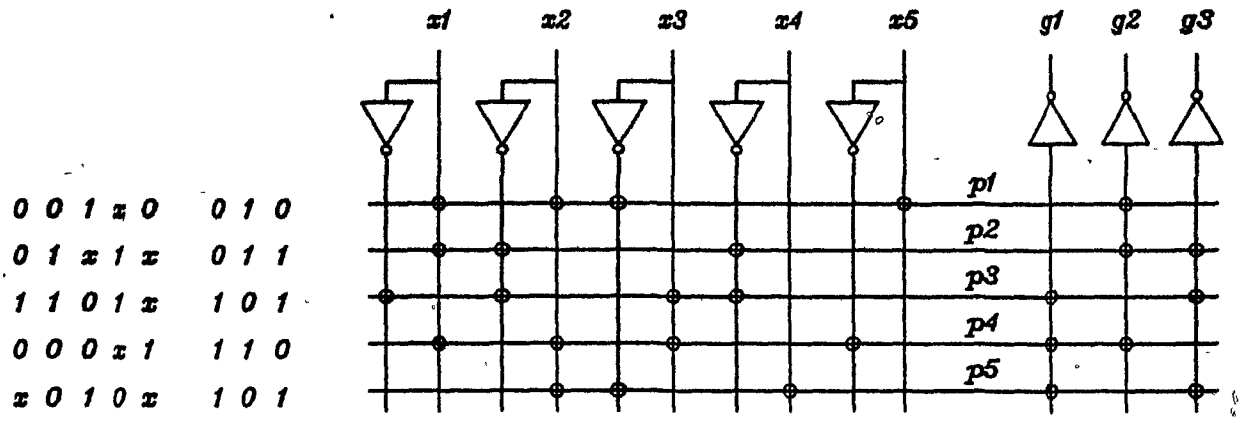


Figure 5.1 Assumed Layout Positioning for NOR-NOR Implementation

path.

For the bridges between wires on the same layer (table 4.1), it is necessary to identify the patterns discussed in section six. The process is similar to the one for additional cross-point faults. In the case of bridging faults, two adjacent bits in the PLA representation have to be observed. It has been assumed that the implementation is NOR-NOR and the layout follows the structure in figure 5.1.

The implementation uses bit-wise operations extensively. The total time required to generate the CSA faults corresponding to cross-point faults, cross-point shorts, and bridging faults (AND and OR effect) for the example PLA's is shown in table 5.2. The set of completely specified CSA faults is subsequently used in PODEM along with the two-level representation of the PLA.

5.2 ATPG Tool

The starting point in modifying the implementation of PODEM is the data structure for the fault dictionary. Each stuck-at fault in the old structure has to be capable of accepting a list of conditions. An image of the new data structure is shown in figure 5.2. Each condition, along with the corresponding stuck-at fault counts as a completely specified CSA fault while the stuck fault itself counts as a CSA fault with null condition. Thus, the fact that many cross-point and bridging faults can be mapped into CSA faults with null condition reduces the final size of the CSA fault set

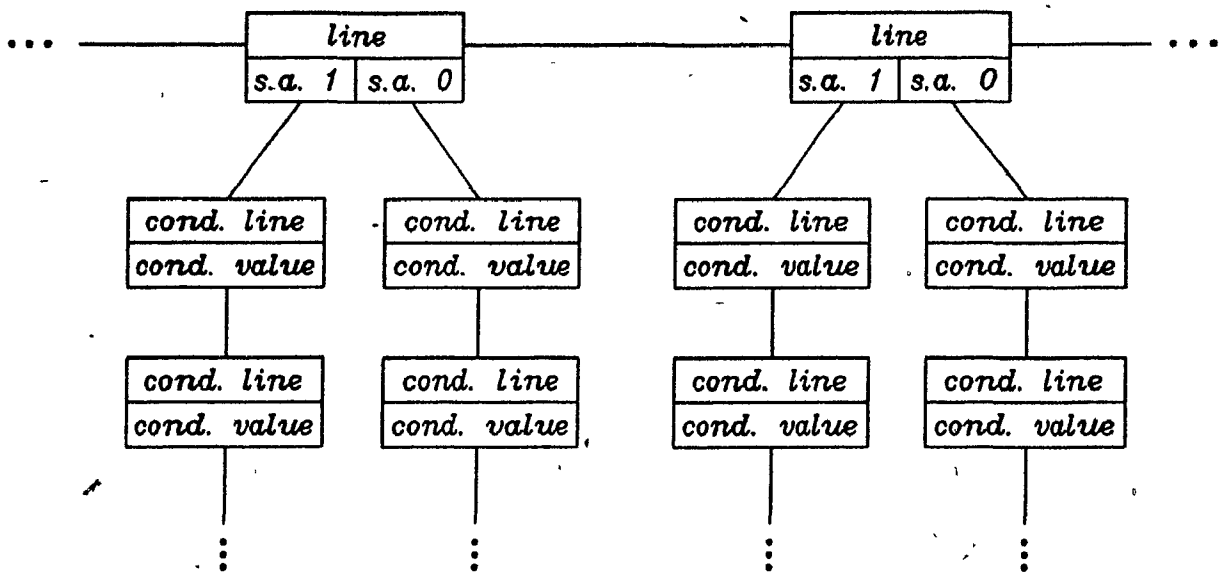


Figure 5.2 Fault Dictionary Data Structure

The general test generation strategy is to consider each fault on each line in the circuit. The CSA faults with completely specified conditions are considered first. If at least one such fault is detectable, then both the completely specified and the null condition CSA fault are counted as detected. If all the completely specified CSA faults attached to some line are undetectable, then the null condition CSA fault is considered alone.

A few statements have been added to the forward implication part of the program such that the value on the condition line is monitored. If the actual value does not match the condition value, then a backtrack operation is performed. If the values match ("don't care" is considered match), the program continues. This represents a third case which require bounding, which is added to the other two cases introduced in [24]. For the case of a "don't care" on the condition line, an extra function was required to specify PLA input values for the cone leading to the condition line. This is necessary because normally, the unspecified inputs are randomly assigned and the test vector is then simulated.

It is important to note that no test set compaction is attempted, other than random assignment of the unspecified inputs and fault simulation. Various PLA-specific heuristics have been presented in [9], [33] for this purpose. The CSA fault model is

primarily intended for embedded PLA's such that test vectors are generated for the entire circuit.

5.3 CSA Coverage

A set of experiments has been performed on 45 benchmarking PLA's, as follows:

Experiment 1 :

A test set has been generated for all single stuck-at faults and for all single cross-point faults modeled as CSA faults. This test set is called T_{CSA-CP}.

Experiment 2 :

A test set has been generated for all single stuck-at faults and all single bridging faults on the same layer (SLB), modeled as CSA faults. This test set is called $T_{CSA-SLB}$. It has been assumed that the logic effect of the bridging faults is the same, either AND or OR, in the entire PLA.

Experiment 3 :

A test set has been generated for all single stuck-at faults and all single bridging faults between wires on different layers, modeled as CSA faults. This test set is called $T_{CSA-CPS}$, where CPS stands for cross-point shorts. The same assumption has been made as in the previous experiment regarding the effect of bridging faults.

Table 5.3 (pages 57 and 58) shows the coverage obtained for cross-point faults. Column 1 identifies PLA's by names used in [9]; columns 2,3, and 4 provide the number of input, product and output lines, respectively; column 5 is the number of additional cross-point faults which are not covered by stuck-at faults in the two-level model of the PLA; column 6 represents the total number of cross-point faults ; column

7 is the coverage of all single cross-point faults by a complete test set developed using our tool; finally, column 8 is the coverage of single cross-point faults by T_{SA}, which will be described in section 5.4.

Stuck-at coverage is 100% for all cases except for the PLA named *dist*, wherein one stuck fault is untestable. Therefore, the stuck-at coverage is not included in the table. For cross-point faults, the coverage obtained is exactly the same as reported in [9]. Therefore, it can be safely concluded that the CSA model provides the same capabilities as the cross-point fault model in terms of the coverage of CP faults. Recall from [9] that less than 100% coverage in column 8 corresponds to those faults which are undetectable.

Table 5.4 (pages 59 and 60) shows the results for bridging faults between wires on the same layer, for AND and OR effect separately. Columns 1,2,3, and 4 show the name and size of the PLA. Column 5 shows the total number of single Same Layer Bridging (SLB) faults for each PLA. Columns 6 and 9 show the number of bridges which are found to be untestable in the preprocessing step, for AND and OR effect, respectively, according to the results shown in table 4.1. Columns 7 and 10 show the additional number of bridges found to be untestable by the ATPG algorithm. Finally, columns 8 and 11 show the coverage of bridging faults, which includes all untestable bridging faults as well.

Table 5.6 (pages 63 and 64) shows the coverage of cross-point shorts (CPS). Column 5 shows the total number of cross-point shorts ; columns 6 and 8 show the coverage of all cross-point shorts using the CSA fault model. Columns 7 and 9 show the coverage over all cross-point shorts of a complete single stuck-at test set, as it will be explained in section 5.4. In this case, the number of completely specified CSA faults is quite large, due to the fact that it is proportional to the number of unused cross-points in the PLA. An interesting case is the AND effect of the CPS. The CSA fault set for cross-point faults is a subset of the CSA fault set for the CPS, AND effect. Due to the fact that in the case of CPS the fault can be propagated on two paths, the coverage is much higher in the case of CPS than in the case of CP faults.

5.4 Coverage by T_{SA}

It has been determined for all the three sets of experiments, the coverage performance of a complete single stuck-at test set generated on the two-level model of the PLA. This has been done by generating test vectors only for the CSA faults with null condition and by simulating the completely specified CSA faults as well. As mentioned in chapter two, coverage by T_{SA} of cross-point and bridging faults has also been considered by Agrawal and Johnson [11].

For the cross-point faults, the results are shown in table 5.3, column 8. Note that the difference in coverage with respect to T_{CSA} ranges between 0% and 38.3% (*x6dn*). For the bridging faults between wires on different layers, the results are shown in table 5.6 , columns 7 and 9, for AND and OR effect, respectively. Table 5.5 (pages 61 and 62) shows the coverage of bridging faults between lines on the same layer by $T_{CSA-SLB}$ (also shown in table 5.4) and by T_{SA}, for OR and AND effects. Column 2 shows the total number of bridging faults. The coverage obtained by $T_{CSA-SLB}$ is shown in columns 3 and 5, for AND and OR effect, respectively. Columns 4 and 6 show the coverage of bridges by T_{SA}. For the AND effect, the difference in coverage ranges between 0 and 4.47 (*wim*). Bridging faults under the assumption of AND effect were also considered in [11], but the coverage by T_{SA} reported in [11] is over bridging faults **and** stuck-at faults. Since the number of bridging faults considered is much smaller than the number of stuck-at faults, the results presented in tables 5.5 and 5.6 give a more accurate image of the coverage performance of T_{SA}.

These results show that for PLA's, the coverage achieved by T_{SA} is unacceptably low in some cases. There are many examples where the drop in the T_{SA} coverage over non stuck-at faults is more than 15% with respect to T_{CSA-CP}, $T_{CSA-SLB}$, or $T_{CSA-CPS}$.

PLA size				Faults		Coverage	
1	2 IP	3 PR	4 OP	5 add. CP	6 total CP	7 T_{CSA-CP} cov.(%)	8 T_{SA} cov.(%)
add6	12	355	4	6258	11005	88.69	78.64
adr4	8	75	5	820	1575	90.54	82.60
alu1	12	19	8	507	608	97.53	81.09
alu2	10	68	8	1289	1904	83.77	71.80
alu3	10	66	8	1222	1848	87.45	77.22
apla	10	25	12	416	800	99.50	91.25
bc0	21	179	11	5904	9487	95.69	76.39
chkn	29	140	7	5761	9100	93.64	67.60
co14	14	14	1	0	406	100.00	100.00
dc1	4	9	7	54	135	94.07	92.60
dc2	8	39	7	429	897	96.55	92.42
dist	8	120	5	935	2520	93.02	91.31
dk17	10	18	11	320	558	99.82	92.83
dk27	8	10	9	165	250	100.00	96.00
dk48	15	21	17	654	987	99.80	94.94
exep	28	109	62	10403	12862	99.39	88.41
f51m	8	76	8	1104	1824	92.22	80.59
gary	15	107	11	2375	4387	94.80	85.68
in0	15	107	11	2375	4387	94.83	85.16
in1	15	104	17	1951	4888	97.96	87.36
in2	19	135	10	3893	6480	96.50	83.93
in3	34	74	29	5897	7178	97.77	80.84
in4	32	212	20	13132	17808	95.40	66.37
in5	24	62	14	2574	3844	98.39	79.55
in6	33	54	23	3822	4806	98.90	80.55
in7	26	54	10	2584	3348	95.31	76.46
jbp	36	122	57	13993	15738	98.48	87.77
misg	56	69	23	8894	9315	97.11	62.52
mlp4	8	127	8	1400	3048	94.69	93.04

Table 5.3 Cross-Point Fault Coverage

PLA size				Faults		Coverage	
1	2 IP	3 PR	4 OP	5 add. CP	6 total CP	7 T_{CSA-CP} cov.(%)	8 T_{SA} cov.(%)
radd	8	75	5	820	1575	90.54	84.43
rckl	32	32	7	1056	2272	100.00	100.00
rd53	5	31	3	88	403	95.28	95.28
rd73	7	127	3	500	2159	92.22	92.22
risc	8	28	31	1002	1316	97.12	94.60
root	8	57	5	524	1197	89.72	86.30
sqr6	6	50	11	676	1150	93.21	92.69
sqn	7	38	3	235	646	91.33	82.66
vg2	25	110	8	4664	6380	95.08	71.55
wim	4	9	7	59	135	94.82	87.41
x1dn	27	110	6	4562	6600	96.15	69.27
x6dn	38	81	5	5112	6561	97.35	59.01
x9dn	27	120	7	4924	7320	95.94	72.68
z4	7	39	4	499	702	85.04	75.08
Z5xp1	7	65	9	933	1495	93.04	88.89
Z9sym	9	84	1	504	1596	100.00	69.55

Table 5.3 Cross-Point Fault Coverage – Continued

PLA size				Faults	AND effect			OR effect		
1	2	3	4	5	6	7	8	9	10	11
	IP	PR	OP	total SLB	nr. untest. pproc.	PODEM	cov. (%)	nr. untest. pproc.	PODEM	cov. (%)
add6	12	355	7	383	0	0	100	345	2	100
adr4	8	75	5	93	0	0	100	65	3	100
alu1	12	19	8	48	0	0	100	12	0	100
alu2	10	68	8	93	0	0	100	55	10	100
alu3	10	66	8	91	0	0	100	59	3	100
apla	10	25	12	54	0	0	100	3	0	100
bc0	21	179	11	229	0	0	100	23	5	100
chkn	29	140	7	202	0	0	100	125	10	100
co14	14	14	1	40	0	0	100	14	0	100
dc1	4	9	7	21	0	0	100	1	0	100
dc2	8	39	7	59	0	0	100	21	5	100
dist	8	120	5	138	0	0	100	90	9	100
dk17	10	18	11	46	0	0	100	1	0	100
dk27	8	10	9	32	0	0	100	1	0	100
dk48	15	21	17	65	0	0	100	5	0	100
exep	30	109	62	224	0	0	100	4	32	100
f51m	8	76	8	97	0	0	100	72	2	100
gary	15	107	11	145	0	0	100	33	19	100
in0	15	107	11	145	0	0	100	41	15	100
in1	15	104	17	148	0	0	100	13	9	100
in2	19	135	10	180	0	0	100	92	15	100
in3	34	74	29	168	0	2	100	28	10	100
in4	32	212	20	293	0	1	100	33	94	100
in5	24	62	14	121	0	0	100	12	3	100
in6	33	54	23	140	0	3	100	8	5	100
in7	26	54	10	113	0	1	100	33	8	100
jbp	36	122	57	248	0	1	100	45	12	100
misg	56	69	23	201	0	5	100	57	1	100
mlp4	8	127	8	148	0	0	100	48	33	100

Table 5.4 Same Layer Bridges

PLA size				Faults	AND effect			OR effect		
1	2	3	4	5	6	7	8	9	10	11
	IP	PR	OP	total SLB	nr. untest. pproc.	PODEM	cov. (%)	nr. untest. pproc.	PODEM	cov. (%)
radd	8	75	5	193	0	0	100	67	5	100
rckl	32	32	7	100	0	0	100	2	0	100
rd53	5	31	3	41	0	0	100	27	0	100
rd73	7	127	3	141	0	0	100	120	4	100
risc	8	28	31	72	0	3	100	2	3	100
root	8	57	5	75	0	0	100	28	12	100
sqn	7	38	3	52	0	0	100	27	0	100
sqr6	6	50	11	70	0	0	100	24	2	100
vg2	25	110	8	165	0	0	100	66	12	100
wim	4	9	7	21	0	1	100	1	1	100
x1dn	27	110	6	167	0	1	100	92	8	100
x6dn	38	81	5	159	0	0	100	25	21	100
x9dn	27	120	7	178	0	2	100	102	21	100
z4	7	59	4	54	0	0	100	51	3	100
Z5xp1	7	65	9	85	0	0	100	26	17	100
Z9sym	9	84	1	100	0	0	100	83	1	100

Table 5.4 Same Layer Bridges – Continued

PLA		AND effect		OR Effect	
1	2	3	4	5	6
	Total SLBF	cov.(%) $T_{CSA-SLB}$	cov.(%) T_{SA}	cov.(%) $T_{CSA-SLB}$	cov.(%) T_{SA}
add6	383	100	100	100	100
adr4	93	100	100	100	98.91
alu1	48	100	100	100	85.44
alu2	93	100	100	100	96.75
alu3	91	100	100	100	96.13
apla	54	100	98.16	100	90.47
bc0	229	100	100	100	95.12
chkn	202	100	100	100	100
co14	40	100	100	100	100
dc1	21	100	100	100	100
dc2	59	100	100	100	100
dist	138	100	99.31	100	99.99
dk17	46	100	100	100	89.00
dk27	32	100	100	100	81.59
dk48	65	100	100	100	84.91
exep	224	100	100	100	85.66
f51m	97	100	100	100	98.98
gary	145	100	100	100	97.87
in0	145	100	100	100	97.87
in1	148	100	100	100	98.50
in2	180	100	100	100	98.39
in3	168	100	98.78	100	89.77
in4	293	100	99.62	100	94.77
in5	121	100	100	100	94.99
in6	140	100	97.88	100	94.84
in7	113	100	99.09	100	91.94
jbp	248	100	99.55	100	93.15
misg	201	100	97.53	100	94.04
mlp4	148	100	100	100	100

Table 5.5 Same Layer Bridges - $T_{CSA-SLB}$ and T_{SA}

PLA		AND effect		OR Effect	
1	2 Total SLBF	3 cov.(%) $T_{CSA-SLB}$	4 cov.(%) T_{SA}	5 cov.(%) $T_{CSA-SLB}$	6 cov.(%) T_{SA}
radd	93	100	100	100	100
rckl	100	100	100	100	100
rd53	41	100	100	100	100
rd73	141	100	100	100	100
risc	72	100	95.86	100	97.03
root	75	100	100	100	100
sqn	52	100	100	100	100
sqr6	70	100	100	100	100
vg2	165	100	100	100	93.28
wim	21	100	95.53	100	75.40
x1dn	167	100	100	100	88.97
x6dn	159	100	98.80	100	93.66
x9dn	178	100	100	100	100
z4	54	100	100	100	98.19
Z5xp1	85	100	100	100	100
Z9sym	100	100	100	100	100

Table 5.5 Same Layer Bridges - $T_{CSA-SLB}$ and T_{SA} - Continued

PLA					AND effect		OR Effect	
1	2 IP	3 PR	4 OP	5 Total CPS	6 cov.(%) $T_{CSA-CPS}$	7 cov.(%) T_{SA}	8 cov.(%) $T_{CSA-CPS}$	9 cov.(%) T_{SA}
add6	12	355	7	11005	100	83.17	99.16	95.09
adr4	8	75	5	1575	100	83.37	99.94	93.27
alu1	12	19	8	608	99.83	88.94	100	98.35
alu2	10	68	8	1904	99.90	78.52	99.94	98.11
alu3	10	66	8	1848	99.89	81.54	99.94	92.48
apla	10	25	12	800	100	96.43	100	76.37
bc0	21	179	11	9487	99.99	93.26	99.20	91.31
chkn	29	140	7	9100	99.99	72.81	99.86	93.92
co14	14	14	1	406	100	95.45	100	100
dc1	4	9	7	135	100	93.33	100	95.55
dc2	8	39	7	897	99.89	90.96	100	90.08
dist	8	120	5	2520	99.96	88.85	99.76	93.33
dk17	10	18	11	558	100	100	100	74.37
dk27	8	10	9	250	99.20	95.24	100	78.79
dk48	15	21	17	987	99.90	97.62	99.89	77.71
exep	28	109	62	12862	99.99	88.19	100	49.22
f51m	8	76	8	1824	99.95	85.84	99.62	92.27
gary	15	107	11	4387	100	86.58	99.59	84.91
in0	15	107	11	4387	100	86.90	99.41	84.70
in1	15	104	17	4888	100	86.20	100	85.42
in2	19	135	10	6480	99.99	83.63	99.99	87.48
in3	34	74	29	7178	99.93	83.74	99.91	78.19
in4	32	212	20	17808	99.98	69.50	100	83.09
in5	24	62	14	3844	100	78.71	100	88.47
in6	33	54	23	4806	100	83.21	99.98	84.71
in7	26	54	10	3348	99.79	82.48	99.77	94.35
jbp	36	122	57	15738	99.98	89.66	99.97	76.51
misg	56	69	23	9315	99.94	75.31	99.97	98.35
mlp4	8	127	8	3048	100	94.27	99.96	89.73

Table 5.6 Cross-Point Shorts

PLA					AND effect		OR Effect	
1	2 IP	3 PR	4 OP	5 Total CPS	6 cov.(%) $T_{CSA-CPS}$	7 cov.(%) T_{SA}	8 cov.(%) $T_{CSA-CPS}$	9 cov.(%) T_{SA}
radd	8	75	5	1575	100	85.35	99.54	93.53
rckl	32	32	7	2272	99.86	98.47	100	94.46
rd53	5	31	3	403	100	90.92	100	91.06
rd73	7	127	3	2159	100	86.64	100	95.79
risc	8	28	31	1316	99.77	94.72	99.84	44.31
root	8	57	5	1197	100	85.90	99.90	90.14
sqn	7	38	3	646	100	82.64	100	95.97
sqr6	6	50	11	1150	99.91	94.95	100	86.65
vg2	25	110	8	6380	100	71.45	100	96.55
wim	4	9	7	135	99.27	85.66	98.39	91.11
x1dn	27	110	6	6600	100	72.44	100	96.68
x6dn	38	81	5	6561	99.94	63.03	100	98.70
x9dn	27	120	7	7320	100	74.56	100	94.40
z4	7	39	4	702	100	77.68	100	89.74
Z5xp1	7	65	9	1495	99.93	90.06	100	90.16
Z9sym	9	84	1	1596	100	68.23	100	100

Table 5.6 Cross-Point Shorts – Continued

Chapter 6 Conclusion

This thesis has introduced the Conditional Stuck-At fault model which can model stuck-at, cross-point, and bridging faults. The new model allows deterministic test pattern generation for all these types of faults, using classical test pattern generation and simulation tools, after minor changes. For the case of embedded PLA's, this allows the use of a unique fault model both for the larger circuit and for the PLA, and test for the embedded PLA stuck-at, cross-point and bridging faults, without requiring BIST, additional controllability/observability, or an extended model of the PLA.

It has been proven that any additional cross-point fault can be modeled as a CSA fault and that any missing cross-point fault can be modeled as a null condition CSA fault. Under the assumption that bridging faults have a definite logic effect, either AND or OR, it has been shown that any bridging fault is covered by two CSA faults, any of them, if detectable, being sufficient for detecting the bridge. Therefore, the use of the CSA fault model allows deterministic test pattern generation not only for stuck-at faults or cross-point faults, but also for bridging faults.

The patterns which may occur in the personality matrices and their effect on the CSA faults modeling bridges have been analyzed, identifying the bridges which need to be modeled as completely specified CSA faults, the bridges which are modeled as null condition CSA faults, and a large number of the undetectable bridges. The algorithm derived from this analysis is linear in the number of cross-points. Based on this analysis, an experimental lower bound on bridging fault coverage by any complete single stuck-at test set can be determined for any PLA.

The experimental results presented in chapter five show that the CSA fault model performs as well as the cross-point fault model in terms of the coverage of cross-point faults and much better in terms of the coverage of bridging faults. Coverage figures for stuck-at, cross-point, and bridging faults are reported for 45 benchmarking PLA's. In all cases it has been assumed that the PLA does not have any memory elements attached. The problem of testing such PLA's implementing finite state machines requires further research.

The model for bridging faults also requires further work. The assumption of a well defined logic effect, used for modeling bridges, may not be true in all cases. To verify this assumption, extensive simulation of large PLA's for various locations of bridges is required.

The test pattern generation and especially the simulation tool have to be optimized. The tool used for the experiments presented in chapter five was derived with a minimum effort from an existing tool, to show the validity of the CSA fault model, the main purpose of this work being to present a new method for PLA test pattern generation. Various heuristics can be used both in the test pattern generation and in the simulation parts.

REFERENCES

[1] P.P. Gelsinger,"Design and Test of the 80386",*IEEE Design & Test of Computers*, pp.42-50, June 1987.

[2] H.H. Chao, S. Ong, M. Tsai, F.W. Shih, K.W. Lewis, J.Y.F. Tang, C.A. Trempel, H.N. Yu, P.E. McCormick, C.V. Davis, A.L. Diamond, T.J. Medve and J.C.L. Hou, "Designing the Micro/370",*IEEE Design & Test of Computers*, pp.32-40, June 1987.

[3] H. Fleisher and L.I. Maissel, "An Introduction to Array Logic", *IBM Journal Research and Development*, vol. 19, pp. 98-109, March 1975.

[4] T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's", *IEEE Transactions On Computers*, vol. C-33, pp.879-894, October 1984.

[5] C.W. Cha, "A Testing Strategy for PLAs", *IEEE Design Automation Conference*, pp.326-330, 1978.

[6] J.E. Smith, "Detection of Faults in Programmable Logic Arrays", *IEEE Transactions On Computers*, vol. C-28, pp.845-853, November 1979.

[7] D.L.Ostapko, S.J. Hong, "Fault Analysis and Fault Generation for Programmable Logic Arrays", *IEEE Transactions On Computers*, vol. C-28, pp.617-626, Sept. 1979.

[8] P.Bose,J.A.Abraham, "Test Generation for Programmable Arrays", *Proceedings of the 19th Design Automation Conference*, pp. 574-580, August 1982.

[9] R.S. Wei and A. Sangiovanni-Vincentelli, "PLATYPUS: A PLA Test Pattern Generation Tool", *Proceedings of the 22th Design Automation Conference*, pp. 197-203, June 1986.

[10] M. Robinson and J. Rajski, "PLANET: A Test Set Generation Program for PLAs", *IEEE Proceedings of the Pacific RIM Conf. on Communications,Computers, and Signal Processing*, pp. 292-295, June, 1987.

[11] V.D.Agrawal and D.D.Johnson, "Logic Modeling of PLA Faults", *IEEE Proceedings of the International Conference on Computer Design*, pp. 86-88, Oct.1986.

[12] W.Maly, "Fault Models for the NMOS Programmable Logic Array", *IEEE Proceedings of Custom Integrated Circuits Conference*, pp.467-470, May 1986.

[13] V.K.Agarwal, "Multiple Fault Detection in Programmable Logic Arays", *IEEE Transactions On Computers*, vol.C-29, pp. 518-522, June 1980.

[14] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Tests", *IEEE Transactions on Computers*, vol. C-30, pp. 823-828, 1981.

[15] S. Bozorgui-Nesbat and E.J. McCluskey, "Lower Overhead Design for Testability of PLA's", *Proceedings of the IEEE International Test Conference*, pp. 856-865, November, 1984.

[16] J. Rajski and V.K. Agarwal, "Teating Properties and Applications of Inverter-Free PLA's", *Proceedings of the IEEE International Test Conference*, pp. 500-507, November, 1985.

[17] K.A. Hua, J.-Y. Jou and J.A. Abraham, "Built-In Tests for VLSI Finite State Machines", *14th International Symposium on Fault Tolerant Computing*, pp. 292-297, 1984.

[18] W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays", *IEEE Transactions on Computers*, vol. C-30, pp. 829-833, 1981.

[19] R. Treuer, *A New Design of Built-In Self Testing Programmable Logic Arrays with High Fault Coverage and Low Overhead*, McGill University, Electrical Engineering Department, Master Thesis, 1985.

[20] R.Treuer,H.Fujiwara,V.K.Agarwal, "Implementing a Built-In Self -Test PLA Design", *IEEE Design and Test of Computers*, pp.37-48, April 1985.

[21] E.B. Eichelberger and T.W Williams, "A Logic Design Structure for LSI Logic", *Proceedings of the 14th Design Automation Conference*, pp. 462-468, June 1987.

[22] P.S. Bottorf, R.E. France, N.H. Garges, E.J. Orosz, "Test Generation for Large Logic Networks", *Proceedings of the 14th Design Automation Conference*, pp. 462-468, June 1987.

[23] J.P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", *IBM Journal Research and Developement*,vol. 10,pp. 278-291,July 1966

[24] P.Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Circuits",*IEEE Transactions on Computers*, Vol. C-30 No.3,pp 215-222, March 1981.

[25] H.Fujiwara and S.Toida, "On the Acceleration of Test Generation Algorithms", *Dig., 13th Annu. Int. Symp. Fault-Tolerant Computing*, Milan, Italy, pp. 98-105, June 1983.

[26] V.K. Agarwal and A.S.F. Fung, "Multiple Fault Testing of Large Circuits by Single Fault Test Sets", *IEEE Transactions on Computers*, Vol. C-30, pp. 855-865, Nov. 1981.

[27] J.L.A. Hughes and E.J. McCluskey, "An Analysis of the Multiple Fault Detection Capabilities of Single Stuck-At Fault Test Sets", *Proceedings of the IEEE International Test Conference*, pp. 52-58, Oct. 1984.

[28] J. Abraham and V.K. Agarwal, "Digital Systems Test Generation", chapter in *Fault Tolerant Computing: Principles and Recent Advances*, edited by D.K. Pradhan, Prentice-Hall, 1986.

[29] S.M. Reddy and D.S. Ha,"On the Design of Testable PLAs", *Proceedings of the 19th Annual Conference on Information Sciences and Systems*, Johns Hopkins University, pp. 80-88, March 1985.

[30] D.S. Ha, S.M. Reddy, "An Experiment on the Size of Fault Detection Tests for Testable PLA's", *Proceedings of the International Conference on Computer-Aided Design*, pp. 151-156, 1986.

[31] H. Fujiwara, "A New PLA Design for Universal Testability", *IEEE Transactions on Computers*, vol. C-33, pp.745-750, 1984

[32] J. Khakbaz, "A Testable PLA Design with Low Overhead and High Fault Coverage", *IEEE Transactions on Computers*, vol. C-33, pp. 743-745, 1984.

[33] J. Salik, B.Underwood, J.Kuban and M.R. Mercer,*An Automatic Test Pattern Generation Algorithm for PLAs*, *IEEE Proceedings of the ICCAD*, pp.152-155, 1986

[34] J.P. Shen,W.Maly, and F.J.Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits" ,*IEEE Design & Test of Computers on Manufacturing Testing*, pp. 13-26, December 1985.

[35] W. Maly, "Optimal Order of the VLSI IC Testing Sequence", *Proceedings of the 23th Design Automation Conference*, pp. 560-566, 1986.

[36] Private communications with D.D. Johnson.

[37] S. Yajima and T. Aramaki, "Autonomously Testable Programmable Logic Arrays", *11th International Symposium on Fault Tolerant Computing*, pp. 41-43, 1981.

[38] M.H. Schulz, E. Trischler, and T. Sarfert, "Socrates : A Highly Efficient Automatic Test Pattern Generation System", *Proceedings of the IEEE International Test Conference*, pp. 1016-1026, September 1987.

[39] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", *Proceedings of ISCAS*, pp. 663-698, 1985.