This is to certify that the

dissertation entitled

DESIGN OF FAULT-TOLERANT
PROGRAMMABLE LOGIC ARRAYS
FOR YIELD ENHANCEMENT

presented by

Tsin-Yuan Chang

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering

Major professor

Date Nov. 6, 1989

DESIGN OF FAULT-TOLERANT PROGRAMMABLE LOGIC ARRAYS FOR YIELD ENHANCEMENT

By

Tsin-Yuan Chang

A DISSERTATION

Submitted to

Michigan State University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1989

DESIGN OF FAULT-TOLERANT PROGRAMMABLE LOGIC ARRAYS FOR YIELD ENHANCEMENT

BY

Tsin-Yuan Chang

Department of Electrical Engineering
Michigan State University

ABSTRACT

The yield (expected percentage of good chips out of a wafer) of integrated circuits (ICs) has always been crucial to the commercial success of their manufacture. The technology of ICs evolved from LSI, VLSI, to ULSI in the past two decades. Multiple layers and scaling techniques make it possible for more than 10^6 transistors to be put into a single chip. However, as the complexity of digital devices increases and geometry shrinks, the probability of having faulty components also increases, thereby lowering the chip yield.

One solution to the low yield problem is to improve manufacturing and testing processes, but it is very costly and quite difficult to implement within a short time. Another practical way is the use of fault-tolerant structures, which has been demonstrated in practice for high density memory chips. The result of fault-tolerant memory design is a reduction in the capital-required level of shippable product, and also that redundancy typically improves yields by 1.5 to 5 times.

Programmable Logic Arrays (PLAs) have the advantages of regular structure, design simplicity, and fast turnaround time. The use of PLAs becomes increasingly

popular for implementing Boolean logic functions and control blocks in the design of integrated circuit. Due to the fact that complex chips (and in particular microprocessors) can be efficiently implemented using PLAs, a trend towards manufacturing larger programmable chips is expected.

In this dissertation, a fault-tolerant design for large PLAs is proposed. The fault-tolerant design achieves a full diagnosability of single and multiple *stuck-at* faults, *bridging* faults, and *crosspoint* faults. During the manufacturing process, faults in the PLA can be detected, located, and repaired with the spare lines. When the PLA is used in field, the structure still possesses the easily testable capability. An automatic layout generator, *MRPLA*, has also been developed and implemented in Sun 3/160 for generating the physical layout of the proposed fault-tolerant PLA. In addition, some important issues such as die size, speed, and yield enhancement are also addressed in this study. The results of this study show that the yield can be enhanced significantly. A simple, yet efficient optimization method has been presented to determine the optimal redundancy of various sizes of PLAs.

This study also introduces a PLA structure based on memory cells. The RAM-Based PLA (RBPLA) allows designers to reprogram the PLA as many times as needed. A fault-tolerant RBPLA is also presented to electrically repair faults in the manufacturing process and also in field use.

ACKNOWLEDGMENTS

The author wishes to express his sincere appreciation to his major advisor, Dr. Chin-Long Wey, for the guidance and encouragement given in the course of this graduate study. He also wishes to thank the dissertation committee members, Dr. Donnie Reinhard, Dr. Michael Shanblatt, and Dr. Byron Drachman for their valuable suggestions and comments in his dissertation research. The author also gives thanks to Dr. Harriett Rigas, and regrets her passing.

The author is especially grateful for the financial support from Dr. Wey and the National Science Foundation under grant No. MIP-8700880. Without these supports, this research effort would not have been possible.

He would like to acknowledge all the faculty and staff members, and the students who gave him help and assistance during his studying in the Electrical Engineering Department at Michigan State University, and many friends, especially from the Church in Lansing, who showed their support and concern.

Finally, he is very grateful to his family for years of concern, encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

As the complexity of digital devices increases and the geometry shrinks, the probability of having faulty components also increases. The yield of integrated circuits (expected percentage of good chips from a wafer) has always been crucial to the commercial success of their manufacture.

One solution to solve low yield problems is to improve manufacturing and testing processes, but it is very costly and quite difficult to implement within a short time [57]. Another practical way is the use of fault-tolerant structures [39], which has been demonstrated in practice for high density memory chips. The only integrated circuits so far to have exploited fault-tolerant techniques commercially have been memory chips. This is because memory chips are particularly densely packed and therefore increasingly vulnerable to defects, and also because a regular memory array lends itself to a variety of efficient fault-tolerant designs. The result of fault-tolerant memory design is a reduction in the capital-required level of shippable product, and also that redundancy typically improves yields by 1.5 to 5 times [56].

During the past few years, Programmable Logic Arrays (PLAs) have become increasingly common for implementing Boolean logic functions in Very Large Scale Integration (VLSI) chips. The advantages of regular structure, design simplicity, and fast turnaround time have played significant roles in the manufacturing of large density PLAs. Due to the fact that complex chips (and in particular microprocessors) can be efficiently implemented using PLAs, a trend towards manufacturing larger

programmable chips is expected. Therefore, the probability of having faulty PLA chips also increases. The same scenario happens in PLA chips as in memory chips. Low yield, then, is a potential problem in manufacturing of PLA chips.

In recent years, research has extensively dealt with the fault detection and test generation of PLAs [6,9,25,33,54,59]. In particular, redundancy techniques have been successfully applied to PLA testing. While extra logic has been added to PLAs to implement function-independent tests [9,59] so that the complexity of PLA test generation can be reduced, other approaches have implemented the added redundancy with coding techniques, such as m-out-of-n codes and Berger codes [33], to design totally self-checking (TSC) PLAs. Little emphasis, however, has been devoted to use redundancy for repairing PLAs.

1.1 Problem Statement

New devices traditionally push against the current technological limits and often have a very low yield. This situation is only sensible when continuing advances in processing techniques are likely to ensure a profitable yield level by the time the device is in volume production.

When a new generation fabrication process is being developed, the rate of climbing the learning curve is relatively slow, and the initial yield values are typically quite low (Curve 1 in Figure 1.1) [57]. The slow learning is due to the following facts: (1) the immature process design and the technology development vehicle are not centered with respect to process variation; and (2) significant yield drops can be experienced even in mature processes, because it may take a long time before the causes are diagnosed and the problems are corrected. The time necessary to bring the yield above the economically acceptable yield (Y_{acc}) can be on the order of several months, resulting in loss in revenue and competitive edge.

The learning curve may be improved by (1) increasing the initial yield; (2) minimizing process and circuit sensitivity to process variation; and (3) characterizing

the most likely failure modes to speed-up diagnosis. As a result, the yield Y_{acc}, as the desired learning curve shown in Curve 2 of Figure 1.1 [57], can be obtained in a shorter period of time. In other words, the yield can be enhanced significantly if the manufacturing process and testing process are improved. However, this improvement requires better factory equipment and better knowledge of design and testing of the chips, which are very costly and quite difficult to implement. Recently, fault-tolerant techniques have been widely applied to the newly developed fabrication processes.

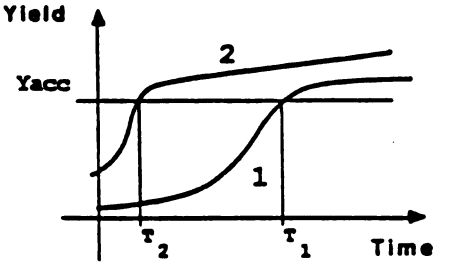


Figure 1.1 Learning Curves [57].

It should be noted that the entire manufacturing process may consist of three major yield steps that affect the total number of functional integrated circuit products that are realized [47]. The major steps are: *wafer processing yield*, *probe yield*, and *final test yield*. *Wafer processing yield* is defined as the percentage of good wafers that survive the manufacturing process. *Probe yield* is defined as the percentage of good chips out of a wafer. *Final test yield* is the percentage of devices that pass a final test program which occurs after the die have been wire bonded to a lead frame and placed inside a package.

Fault-tolerant techniques have been used extensively by semiconductor manufacturers [39]. The use of redundancy for yield enhancement is not new; the first scheme for redundancy implementation on core memories was published in 1964 [44]

and the first practical application in redundant memory design was proposed in 1979 [7]. Currently, more than 15 semiconductor memory manufacturers have commercially produced various redundant memory chips [27,39]. As the technique of the fault-tolerant memory design matures, the next logical step is to apply this technique to PLAs.

1.2 Objectives

The motivation for incorporating fault-tolerance into PLAs is twofold: yield enhancement in the manufacturing phase and fault-tolerance in field. Both are achieved by restructuring the links so as to isolate the faulty lines.

This work is centered on the study of the design of fault-tolerant PLAs. The issues include *design-for-repairability*, *design-for-diagnosability*, and *design-for-manufacturability/yield*.

Design-for-repairability is the design of a repairable PLA that implements a reconfiguration scheme to replace faulty lines by spare lines. *Reconfiguration* is defined as an operation for replacing faulty components with spares while maintaining the original interconnection structure.

Before the partially defective PLA chips can be repaired, the types and locations of faults must be precisely identified, so that the repair process can be properly and efficiently performed. The need for locating and identifying faults led to the *design-for-diagnosability*.

Design-for-manufacturability/yield is aimed at achieving manufacturable, high-yield chips. To enhance chip yield of PLAs, spare lines and reconfiguration circuitry are built into the chip so that partially defective chips can be repaired. Since spare lines and reconfiguration circuitry are also susceptible to defects, too much redundancy may have a "diminishing return" effect on the chip. Therefore, the amount of additional redundancy to the PLA is best kept as low as possible. However, if the amount of redundancy is insufficient, high yield cannot be reached.

Two aspects of fault-tolerance can be identified: (1) techniques to tolerate manufacturing defects; and (2) techniques to tolerate failures in field. In this work, the fault-tolerant PLA designs that fulfill the above design aspects are investigated. Fault-tolerant PLA design using laser programming techniques is implemented to tolerate the manufacturing faults, while fault-tolerant RAM-based PLA design is implemented using electrical programming techniques to tolerate manufacturing defects and to tolerate failures in field.

1.3 Physical Failures in VLSI Circuits

VLSI systems have the following two classes of failures [60]: *manufacturing failures* and *long-term failures*. *Manufacturing failures* are caused by defects which depend on the processes and materials; while *long-term failures* are caused by wear-out in field. *Long-term failure* mechanisms include break-in lines, shorts between lines, and degradation or breakdown of active devices.

The manufacturing defects can be divided into two groups: those that affect a relatively large (global) area of the wafer and those that affect a relatively small (local) area [48]. Examples of global defects include cracks or scratches in the material, photolithographic mask misalignment, line dislocations, and major fabrication process control errors. These defects usually have global and prominent effects on the circuit behavior and can be detected easily early in the manufacturing phase. Furthermore, for a finely tuned and mature fabrication line, major processes control errors, and hence global defects, can be detected easily and minimized. For the above reasons, localized spot or point defects are the primary targets for fabrication testing.

Point defects can be classified into three categories: silicon substrate inhomogeneities, local surface contaminations, and photolithography-related point defects. The origin of defects from each of these categories involves distinct, usually complicated and frequently uncontrollable processes. Complete and accurate physical modeling of point defects inherent in the fabrication process is difficult [48].

Depending on the location, size, and type, a defect may or may not have any effect on the circuit. Only those significant defects which cause faults are considered in causing faults. For example [48], Figure 1.2 shows that a small point defect in the implant window of a depletion mode MOS transistor may or may not have any significant effect at the location. Figure 1.3 illustrates how a missing element of a polysilicon path may or may not have any significant effect at the circuit level.

Point defects will cause extra or missing spots of metal, polysilicon, or diffusion layouts. Extra spots may cause shorts between two layers (metal, polysilicon, or diffusion), degradation of elements, or extra devices. On the other hand, missing spots may cause break of a line (metal, polysilicon, or diffusion line), degradation of elements, or missing devices.

Fault models are extracted from significant physical failures, and serve two purposes: test generation and fault coverage evaluation. A good fault model is one that is simple to analyze and yet closely represents the behavior of physical faults.

1.4 Redundancy Architectures

The important criteria for evaluating a reconfiguration scheme are hardware overhead, reconfiguration effectiveness (the probability that an array with a given number of faulty cells is reconfigurable), wiring length after reconfiguration, time required for the reconfiguration procedure, and overall yield and reliability [61].

The redundant designs of VLSI array structures can be classified by the following four reconfiguration schemes [61]: (1) the whole row (and/or column) bypass (WRB/WCB); (2) single-cell bypass (SCB); (3) interstitial scheme; and (4) duplicated cell scheme.

The first scheme allows for a faulty cell to cause the whole row or column to be bypassed as shown in Figure 1.4 (a). The control circuitry in the WRB/WCB scheme is simpler than those in others. However, the utilization of spare cells is inefficient. To choose the minimum number of spare rows and/or columns that cover all the faulty

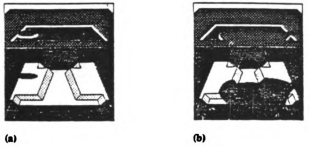


Figure 1.2 Implant Mask Defects [48]:
(a) Non-significant Defect; (b) Significant Effect.

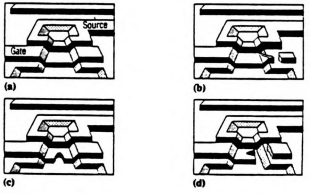


Figure 1.3 Significant and Insignificant Defects [48]:
(a) Defect-free Poly Path; (b) Insignificant Missing Poly;
(c) Insignificant Missing Poly; (d) Significant Missing Poly.

cells is an NP-complete problem [30]. This leads to various heuristic reconfiguration algorithms that have been proposed [5,13,20,58,63].

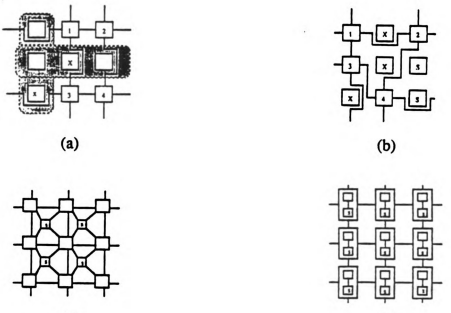


Figure 1.4 Reconfiguration Architectures [61]:
(a) WCB/WRB Scheme; (b) SCB Scheme;
(c) Interstitial Scheme; (d) Duplicated Cell Scheme.

To increase the utilization rate of spare cells, the single-cell bypass (SCB) scheme, as shown in Figure 1.4 (b), allows the faulty cells to be passed. However, the utilization rate is dependent on the complexity of the control circuits that include switches and interconnecting wires. As a result, long interconnection wires after reconfiguration are possible if higher utilization is attained.

The interstitial scheme, as shown in Figure 1.4 (c), has spare cells uniformly distributed into the array and a faulty cell that can only be replaced by its neighboring spare cells. Since spare cells are adjacent to regular cells, the length of connecting wires is limited, which thus results in a low time overhead. However, the drawback

is that an array may fail due to the lack of spare cells in one area, while there are unused spares in other area.

The last scheme, indicated in Figure 1.4 (d), is the duplicated cell scheme in which each regular cell has its own spare cell. It requires a simple reconfiguration algorithm and low time overhead, but the area overhead is large.

Since each cell of the array in both memories and PLAs takes a very small portion of the entire array, the use of scheme (2)-(4) that requires either high complexity of control circuit, or high percentage of area overhead, is not practical. In this study, the WRB/WCB scheme is implemented in the design of fault-tolerant PLA. That is, the faulty lines are repaired and replaced by the spare lines.

1.5 Thesis Organization

This dissertation is organized as follows. Chapter 2 reviews the design of fault-tolerant semiconductor memories. Two commercial memory chips that implement the fault-tolerant design are presented. The laser programming techniques developed in these two examples can be applied to the proposed fault-tolerant PLA design.

In Chapter 3, a fault-tolerant design of PLAs is proposed. The fault-tolerant design achieves a full diagnosability of single and multiple *stuck-at* faults, *bridging* faults, and *crosspoint* faults. During the manufacturing process, faults in the PLA can be detected, located, and repaired with the spare lines. When the PLAs are used in field, the structure still possesses the easily testable capability. In addition to the fault-tolerant structure, an automatic layout generator, called *MRPLA*, is presented to generate the physical layout of the proposed fault-tolerant design. Some important issues in a redundant design, such as chip area and propagation delay time, are also addressed.

Chapter 4 describes the fault diagnosis and repair process for the proposed fault-tolerant PLA. Two examples will be given to demonstrate that the proposed

fault-diagnosable PLA achieves a full diagnosability. In addition, a simple test process is presented for detecting faults after the chip is packaged and used in field.

Chapter 5 analyzes the effects of adding redundancy to the design of fault-tolerant PLAs. A yield model for this design is presented and simulated. Based on the yield model a simple, yet efficient optimization method is proposed to determine the optimal redundancy of various sizes of PLAs.

Chapter 6 illustrates a RAM-based PLA (RBPLA) structure that allows the designers to change the design as many times as needed. In addition, a fault-tolerant design of the RBPLA is also presented. Faults occurred in either the manufacturing process or in field use can be detected, located, and repaired.

Finally, the last chapter summarizes the work of this dissertation research and presents suggestions for related future research.

CHAPTER 2

Fault-Tolerant Semiconductor Memories

Semiconductor memory has made tremendous contributions to the revolutionary growth of digital electronics. The cost and space effectiveness of MOS DRAMs (Dynamic Random Access Memories) has permitted their use in today's computers, for example, more than 100M bytes for mainframe and even 1M bytes for personal computers. MOS SRAM (Static RAM), with low stand-by power, has been used in small, portable, battery-backed systems [4]. Nonvolatile memories such as EPROMs (Electrically Programmable Read-Only Memories) and EEPROMs (Electrically Erasable PROMs) have opened up new areas of applications such as field-programmable microcomputers. Various needs from different systems applications constitute the driving force toward improved performance/cost and enhanced functions of semiconductor memories.

Throughout the short history of semiconductor memories, the number of memory cells in a device has quadrupled approximately every four years [4]. The device density has been increased from 64K, 256K, to 1M, and will soon to 4M and 16M in market. As device density has increased, improved design and fabrication methods have been introduced to maintain an adequate yield of good devices per wafer.

On-chip redundancy techniques have been commercially used to eliminate the large number of chip failures due to the local defects, and offered yield improvement in the manufacturing of the commercial memory chips [39]. The result is a reduction in capital required for wafer fabrication to achieve a desired level of shippable product. Instead of 1% or 2% of good dice per wafer in early chip yields, the right combination

of spare bits per die can suddenly make half the wafer good [42]. Basically, on-chip redundancy techniques take extra memory cells as spares. Each device is tested at wafer probe, and if non-functional cells are found, the device is repaired by replacing the non-functional cells with the spares. One of the biggest controversies surrounding on-chip redundancy is whether to make the replacements by blowing fuses electrically or by laser techniques. The argument will be discussed in Section 2.2.

Before the defective memory cells can be repaired, techniques for diagnosing the location of the defective cells and efficient spare allocation strategies are needed. The repair of the on-chip redundancy is generally divided into two phases: diagnosis to detect and locate all faulty cells, and repair to allocate spares for all faulty cells. In Section 2.3, existing fault analysis and repair algorithms are reviewed.

Finally, two commercial memory products with on-chip redundancy are illustrated in Section 2.4. They are: the fault-tolerant 64K DRAM developed by Bell Laboratories [7], and the 8K × 8 high-performance CMOS SRAM developed by Hitachi Ltd., Japan [38].

2.1 On-Chip Redundancy

The only integrated circuits so far to have exploited on-chip redundancy techniques commercially have been memory chips.

At the level of 64K devices, devices are beginning to appear with on-chip redundancy to increase yields and maintain reasonable manufacturing costs. As memory density increases and geometries shrink (via device scaling and circuit innovations) the die size must remain constant for producibility and yield considerations. As such, defect density becomes a much more important factor than with lower density devices since a single defect can wipe out a major section of memory. Process cleanliness becomes more stringent as well [27]. To offset this, on-chip redundancy allows the defective memory cells to be replaced by the spare

cells. Redundancy will become more important and probably mandatory at 256K DRAM level and even higher level of device density. Table 2.1 summarizes the memories which utilize on-chip redundant circuitry [27]. The on-chip redundancy has been commercially implemented to 64K and 256K DRAMs, 16K, 32K, and 64K SRAMs, and some others. The table also shows that the number of spares is relatively small comparing with the device size.

On a memory with redundancy, incoming addresses are compared with the locations of faulty bits; when a match is found, spare bits take over. Substitutions can be made for individual bits, small clusters or large blocks, or rows or columns. Spare rows and columns have become the most popular approach because they represent a reasonable trade-off between yield enhancement and the number of required elements and associated circuitry.

On-chip redundancy techniques are not free of penalties: spare elements increase the chip area, and result in performance degradation and productivity loss. The important attributes for on-chip redundant circuit design are: how much redundancy to employ; how to apply it; and how much it will affect performance, die size, and yield. The number of spare rows or columns is subject to several considerations since spare cells in any form inflate die size and reduce the number of chips per wafer. Furthermore, each spare element demands extra support circuitry which cannot be repaired. Consequently, too much redundancy reduces overall repair efficiency. The yield improvement factor, the ratio of the yield with redundancy to that without redundancy, can be plotted as a function of the yield without redundancy for different number of spare elements. As more spares are added, the curve is eventually increased and reaches a point of diminishing returns. Further increases in the number of spare elements will start reducing the yield improvement factor.

2.2 Repair Techniques

As mentioned previously, the number of programming elements required is an important consideration in the final choice of optimal number of spare elements. On-

Table 2.1 Memories Built with Redundancy [27]

MANUFACTURER	TYPE OF MEMORY	P/N	TYPE OF REDUNDANCY	PROGRAMMING TECHNIQUES	NOTE
OKI ELECTRIC	256K DRAM	MSM37256	6K CELLS		
SIEMENS AG	256K DRAM	NCA	SPARE ROWS & COLUMNS	HIGH VOLTAGE PULSES AT WAFER TEST POLY FUSE	
NTT MUSASHINO	256K DRAM	NCA	SPARE ROWS & COLUMNS	MODE REGISTER	
BELL LABS	256K DRAM	NCA	8 SPARE ROWS & 8 SPARE COLUMNS	POLYSILICON LASER FUSE	
HITACHI	256K DRAM	NCA	1 SPARE ROW & 1 SPARE COLUMN	HIGH VOLTAGE, POLYSILICON FUSE	
IBM	288K DRAM (32Kx9)	NCA	1152 BITS WITH 4 WORD LINES PER CHIP	UNKNOWN	
IBM	72K BIPOLAR DRAM (8Kx9)	NCA	10TH BIT	UNKNOWN	
BELL LABS	64K DRAM	NCA	8 SPARE ROWS & 8 COLUMNS	POLYSILICON LASER FUSES	REQUIRES CRITICAL MECHANICAL POSITIONING
IBM	64K DRAM	NCA			
INMOS	64K DRAM	IMS2600	8 SPARE ROWS & 8 COLUMNS	HIGH VOLTAGE(12V) PULSES AT WAFER SORT, POLY FUSE	REQUIRES MAIN DE-INHIBITING-EXTRA GATE DELAY
INTEL	64K DRAM	I2164	4 SPARE ROWS 4 SPARE COLUMNS	HIGH VOLTAGE PULSES AT WAFER SORT, POLY FUSE	(SAME AS ABOVE)
MOSTEK	64K DRAM	MK4164	8 SPARE COLUMNS	LASER PULSE AT WAFER SORT POLY FUSE	
HITACHI	64K SRAM	NCA	2 SPARE COLUMNS	LASER ZAP, POLY FUSE	
TOSHIBA	64K SRAM	NCA	2 SPARE ROWS	LASER ZAP, POLY FUSE	
TOSHIBA	64K SRAM	NCA	1 SPARE ROW, 2 SPARE COLUMNS	LASER ZAP, POLY FUSE	
TOSHIBA	64K SRAM(CMOS)	TC5564P	2 SPARE ROWS & 1 SPARE COLUMN	LASER PULSE AT WAFER SORT, POLY FUSE	
TOSHIBA	64K SRAM(NMOS)	TC5565P	2 SPARE COLUMNS	LASER PULSE AT WAFER SORT, POLY FUSE	
INTEL	32K SRAM	NCA	6 SPARE ROWS, 4 SPARE COLUMNS	LASER PULSE AT WAFER SORT, POLY FUSE	
INTEL	16K SRAM	NCA	2 SPARE ROWS, 4 SPARE COLUMNS	LASER PULSE AT WAFER SORT, POLY FUSE	
INTEL	16K SRAM	I2167	3 SPARE ROWS	HIGH VOLTAGE PULSES AT WAFER SORT, POLY FUSE	
INMOS	16K SRAM	IMS1400	2 SPARE COLUMNS	HIGH VOLTAGE PULSES AT WAFER SORT, POLY FUSES	
INMOS	16K SRAM	IMS1420/1421	8 SPARE COLUMNS	HIGH VOLTAGE PULSES AT WAFER SORT, POLY FUSES	
MOSTEK	16K SRAM	MK4167	4 SPARE COLUMNS	LASER PULSE AT WAFER SORT	
INTEL	128K EEPROM	NCA	4 SPARE ROWS=128 BYTES	HIGH VOLTAGE PULSE AT WAFER SORT, POLY FUSES	
MOSTEK	64K EPROM	MK2764	25' REDUNDANT MEMORY MATRIX- 2 COLUMNS	HIGH VOLTAGE(25V) PULSES AT WAFER SORT, POLY FUSE	
SEEQ	16K EEPROM	5213	6 SPARE ROWS	EPROM FUSES	
INTEL	32K BIPOLAR PROM	3632	4 SPARE ROWS	UNKNOWN	
MOTOROLA	16K BIPOLAR PROM	MCM76161	ROWS AND COLUMNS	UNKNOWN	
NTT MUSASHINO	4MBYTE ROM[3]	NCA	COMPLETE ROM REDUNDANCY FOUR 1Mb MODULES		

(1) It is said that redundancy increases the yield by a factor of 5 to 30 depending on the maturity of the Fab line.

(2) Most Magnetic Bubble Memories also use redundancy to increase manufacturing yields by means of a "boot loop".

(3) See "A 4Mb Full Wafer ROM" by M.Y. Kitano et al, 1980 IEEE ISSCC Digest of Technical Papers, Feb. 13-15, 1980.

NCA = NOT COMMERCIALLY AVAILABLE

chip storage of the information that identifies the defective cell locations is a key issue in redundancy. The programming elements used for this purpose fall into two categories [53]: the laser programmable and the electrical programmable. Table 2.2 lists a comparison between laser programming and electrically fusible links.

Table 2.2 The Comparison between Laser and Electrical Programming [53]

Feature	Laser Approach	Electrical Fuses
Circuit Layout	Links are placed anywhere	Links must be accessible to external drives via bonding pads or additional on-chip circuitry
Performance	Access time of programmed and nonprogrammed devices are indistinguishable	Speed is generally adversely affected, particularly if both row and column redundancy are used.
Reliability	Since exploded links are covered with final nitride passivation layer, reliability is extremely high	High reliability requires guard rings around link regions
Area Penalty	Area increase for redundancy is slight -- increase will scale down with finer design rules in future devices	Area increase is also slight, but may not scale down as easily because of layout and reliability concerns
Flexibility	Performance margins are easily tailored with "quick fixes"	Layout is not adaptable to unforeseen circuit needs
Equipment costs	Software development requirements and hence costs are large	Initial costs are lower due to relaxed software demands

The major advantages of electrical fuse blowing are that the redundancy can be implemented with minimal initial capital expenditures and that existing test equipment may be used [1]. Also, electrical fuses offer the simplicity of using an unmodified wafer sort machine, but at the cost of requiring each fuse to be connected to a bulky driver transistor. This extra transistor costs area and limits the number of fuses that can be used, thereby complicating the circuit design. Electrical fuses could conceivably be blown inside the memory's package, opening up the possibility of field repair. Laser programming presents the obvious advantage of conserving valuable silicon real estate by eliminating the circuitry associated with blowing electrical fuses, but it requires the addition of a costly laser to the sort equipment, and precise alignment as well. However, lasers allow a wider choice of potential fuse materials.

Trade-offs between ease of design implementation, up-front capital investment, and final product cost determine whether laser or electrical programming is best for a particular memory product [1]. As it is shown in Table 2.1, most of redundant memory designs implement the laser programming techniques that perform "cut" and "patch" operations in polysilicon links or fuses. Recently, Sandia National Laboratories [3] have also devised a speedy method of on-chip repair that uses low power lasers to cut and patch the metal lines.

2.3 Fault Analysis

Before the repair process is performed, fault analysis algorithm is first called upon to determine if there are any catastrophic problems on the chip or more defects than the spare elements. If not, faults are further analyzed, and the sites of faulty cells are logged into a fault map. According to the fault map, an efficient spare allocation of redundant rows and columns is applied to provide the repair solution.

The problem of spare allocation of redundant elements can be specified as follows. Consider a rectangular array that consists of $M \times N$ cells, as shown in Figure 2.1, where the dots in the array represent the faulty cells, and 2 spare rows (S_r=2) and 3 spare columns (S_c=3) are assigned.

A partially defective chip is said to be *repairable* if the spare elements can completely cover all faulty cells; otherwise, it is *unrepairable*. Therefore, the objective of the fault analysis algorithm is either to quickly check the unrepairability, or to provide repair solutions for the repairable devices. More specifically, if the unrepairability of a device can be quickly determined, then the costly repair process can be terminated early. On the other hand, if spare elements can be efficiently utilized to cover the faults, then more devices can be claimed as good ones.

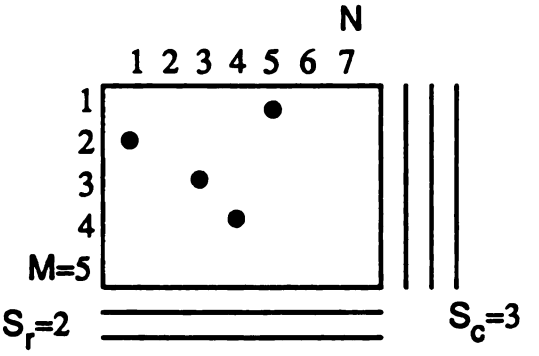


Figure 2.1 Spare Allocation of Redundant Elements.

The problem of optimal spare allocation has been shown as an NP-complete problem [30]. As a result, several heuristic algorithms have been proposed and they are summarized in [20,58,63]. These heuristic algorithms can be classified into two categories: row/column selection and unrepairability checking.

The following heuristics are used to select rows or columns for repair: *Broadside* [58], *Repair-most* [58], and *fault-driven* [13].

The *broadside* approach employs a crude technique to locate each faulty bit and to immediately repair it. No optimization is used. Spares are allocated in a very inefficient fashion, since no overall distribution of faulty bits is considered. This results in failure to identify a potentially repairable device.

A limited usage of optimization techniques can be found in *repair-most* [58]. In this technique, row and column fault counts are employed to determine spare allocations. *Repair-most* is implemented in a two-stage algorithm: *must-repair* and final repair. *Must-repair* determines either a row or a column that must be replaced by a fault-free spare to repair the maximum number of faulty bits. This process is iteratively repeated until no more faulty bits are left uncovered in memory by using spares. This corresponds to a maximization criterion in fault selection; a minimization in allocation of spares can be accomplished by an initial covering of faulty bits. This information is supplied to final-repair to find a balanced time allocation for the desired

repair solution. This is accomplished by considering processing time, laser repair time, and spare utilization [58]. Although this approach gives better results than the broadside approach, undesirable features, such as an inability to provide repair solutions for certain devices and no provision for user-defined preferences, are still left.

Fault-driven [13] partially avoids the drawbacks of the *repair-most* approach. In *fault-driven*, repair solutions are generated according to user-defined preferences. Repair is implemented using a two stage analysis: *forced-repair* and *sparse-repair*. Fault-counters are still employed. *Forced-repair* determines specific rows or columns that must be replaced by redundant copies; *sparse-repair* determines repair solutions for all remaining faulty bits at completion of forced-repair.

The following heuristics are used to check the unrepairability: *Diagonal-test* [5], *Maximum-matching* [30], *Total-faults* [20], *Fault-count after Must-repair* [58], and *Leading-element-test* [63].

The *diagonal-test* approach is a fast test performed on the bits along the major diagonal line of the memory. Since all the faulty bits on a diagonal line of a memory cannot be repaired by the same row or column, if the number of faulty bits on a diagonal line is greater than the total number of spare rows and columns, the memory is unrepairable.

Maximum-matching approach uses the aid of graph theory. If the size of the solution found in the graph is greater than the total number of spare rows and columns, the memory is unrepairable.

Total-faults approach exploits the fact that the maximum number of faulty bits that can be repaired by S_r spare rows and S_c spare columns is $M \times S_c + N \times S_r - S_c \times S_r$. If the number of faulty bits in the memory is greater than that maximum number, it is unrepairable.

Fault-count after Must-repair approach indicates that the total number of unrepaired faults which can be recovered after *Must-repair* is complete is $2 \times S_c \times S_r$,

since there can be no more than S_c faults on a row, and no more than S_r faults on a column.

Leading-element-test approach finds the first faulty bit in the row of the memory as the *Leading-element*. Let |S| be the total number of leading elements, and d be the number of leading elements in which the other non-leading-element faulty bits appeared at both the same row and column. If |S| or |S|+d is greater than the total number of spare rows and columns, the memory is unrepairable.

Recently, several other fault-analysis algorithms [8,22,23,24] have also been proposed to efficiently determine the repair solutions.

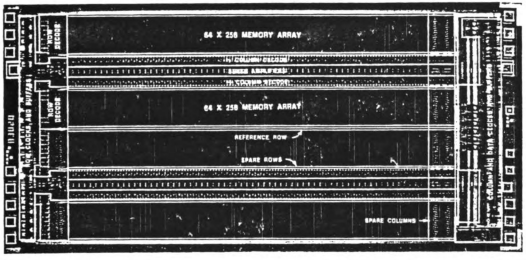
2.4 Fault-Tolerant RAM Design Examples

This section describes two fault-tolerant memory designs that have been commercially available.

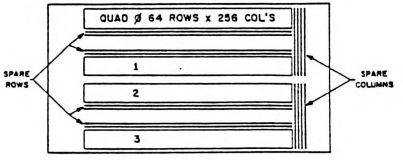
2.4.1 A Fault-Tolerant Dynamic RAM

Figure 2.2 shows a fault-tolerant 64K DRAM developed in 1979 by Bell Laboratories [7]. The design employs a total of 16 spare elements, 8 spare rows and 8 spare columns. Two spare rows, complete with decoder and driver circuitry, are associated with each 16K quadrant, organized as 64 rows by 256 columns. Either one of these spare rows may replace any one of the 64 main rows in the adjacent quadrant or may replace each other, if necessary. Four spare columns including decoder and sense amplifier circuits, are associated with each pair of 16K memory quadrants. Any one of the spare columns may be used to replace any of the 256 columns in the adjacent quadrant or any other previously encoded spare column in the same group.

Replacement of a defective memory element, whether row or column, may be understood by referring to Figure 2.3, which shows a standard and a spare row



(a)



(b)

Figure 2.2 Fault-Tolerant 64K DRAM [7,52]:
(a) Physical Layout; and (b) Schematic Diagram.

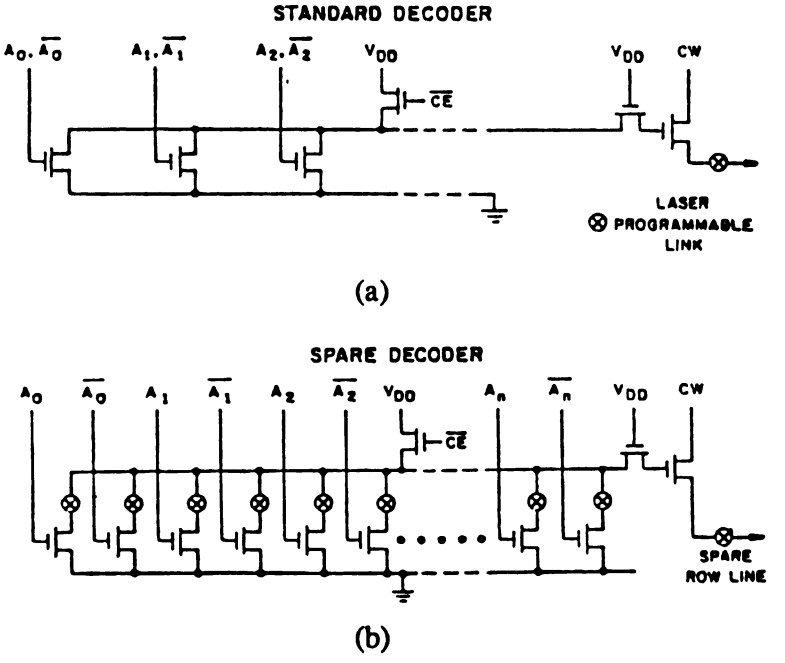


Figure 2.3 Standard and Spare Row Decoders [52]:
(a) Standard Row Decoder; and (b) Spare Row Decoder.

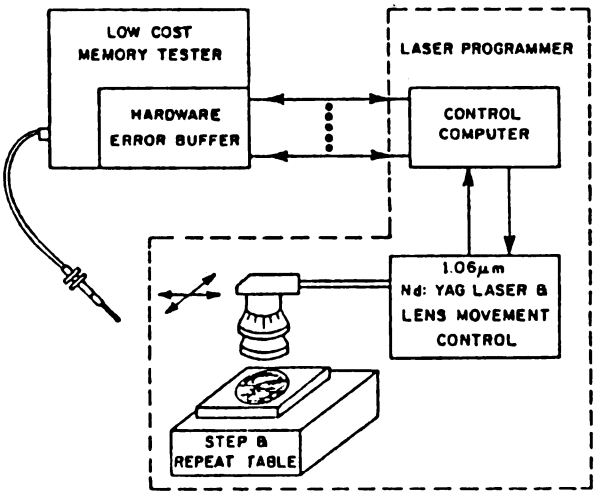


Figure 2.4 Block Diagram of Major Hardware Components of Laser Programming System [52].

decoder schematic. The essential difference between the standard and spare decoder is that the former has half the number of decode transistors of the latter. The identity of the standard decoder is defined by unique connections of address and address complement to appropriate decode gates. By contrast, the spare decoder has both address and complement address tied to the gates of decode transistor pairs.

Disconnection of a faulty memory row is accomplished by exploding the programmable link between the standard row driver and the row line, using a single laser pulse.

An "open" link is defined as one having an impedance of greater than 10 M Ω, and this requirement is easily accomplished in practice. The links are 3-μm polysilicon lines, deposited and patterned along with all active transistor gates and covered with phosphorus glass, as is the balance of the chip prior to metallization. No separate processing steps are associated with the polysilicon links.

Link opening is done on a commercially available laser trimmer which has been modified to include improved positioning accuracy and a TV camera for visual monitoring and alignment. Functional testing and laser programming are fully automatic and require no additional wafer handling or manual intervention. Replacement of each defective row or column takes about 1 s. A chip with no defective bits requires no laser opening.

Figure 2.4 [52] illustrates a block diagram showing interconnection between major hardware components of laser programming system. The experiment shows that the programming time is quite short in the total laser appliance, includes chip alignment, target detection, and laser movement.

2.4.2 A Fault-Tolerant Static RAM

Figure 2.5 shows a block diagram of a 8K × 8 high-performance CMOS Static RAM (Hi-CMOS SRAM) developed by Hitachi Ltd., Japan [38]. The SRAM is fabricated using double polysilicon technology to reduce the memory cell size. The

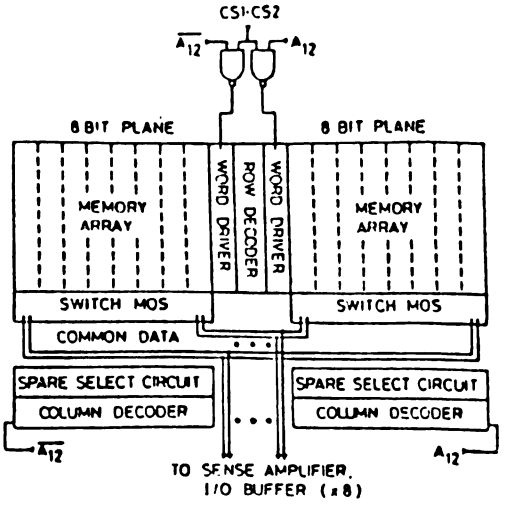


Figure 2.5 Block Diagram of the 8K × 8 Bit Static RAM [38].

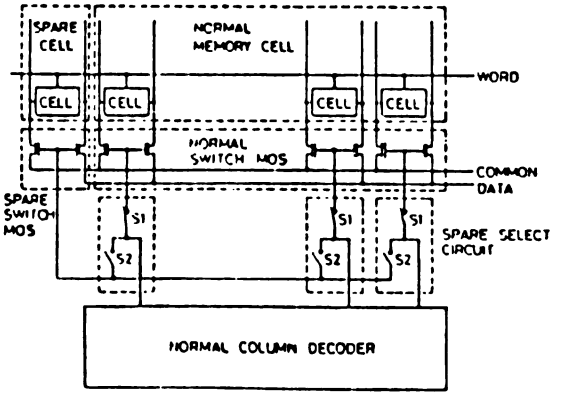


Figure 2.6 Block Diagram of the Redundancy Control Circuit [38].

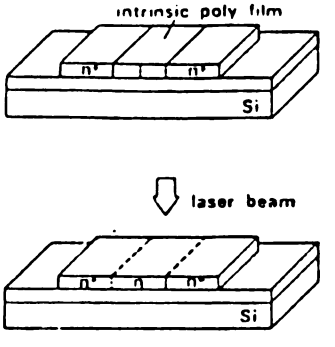


Figure 2.7 Laser Diffusion Programmable Devices [38].

first polysilicon layer is used for the gates of the transistors, while the second layer is for the power supply line. The gate polysilicon length was 2 μm.

In order to improve the manufacturing yield of the SRAM, spare select circuits are added to the column decoder, as shown in Figure 2.6. The redundancy circuit utilizes new programmable devices for the column of the SRAM. The on chip programming is achieved by applying laser pulse to an intrinsic polysilicon film having an n^+ diffusion on either side as shown in Figure 2.7 [38]. Before application of the laser pulses, resistivity between these two n^+ layers is on the order of 10^{10} Ω. When a laser beam is applied to this structure, diffusion takes place from both sides, and the intrinsic part is converted to an n-type. The resistance drops down to 2K Ω. This results in electrical conduction between these n^+ layers. In this design, an N_2 laser-pumped dye laser (wavelength: 0.51 μm, pulse width: 7 ns) with a beam energy of about 10^7 W/cm^2 was applied to intrinsic polysilicon having a 2 μm linewidth and a 4 μm intrinsic layer length. In other words, the laser diffusion programmable device normally stays at the OFF state until it is programmed to the ON state. This programmable device is referred to Normal-off link.

The key advantages of this redundant circuit that utilizes laser diffusion programmable devices are [38]: (1) this technique has excellent compatibility with the Hi-CMOS process; (2) the column spare select scheme causes no access time delay, and features a brief programming time where only two programmable devices per spare are needed; (3) only a very small number of transistors per programming circuit are necessary. Therefore, programming circuit area is minimal; and (4) no damage to surface passivation insulators was observed. This means that the RAM's reliability cannot be affected without the use of extra surface passivation films after the laser process.

2.5 Discussion and Summary

On-chip redundancy techniques have been used extensively by semiconductor manufacturers for fault-tolerant memory designs to enhance the chip yield. This chapter has reviewed the repair techniques and repair process of fault-tolerant RAM designs. Faults in a partially defective memory can be detected, located, and repaired. Due to the similarity of both memory and PLA, the repair techniques and processes developed for fault-tolerant memories should be able to apply to PLAs.

In the next chapter, the design of fault-tolerant PLA is discussed. Faults in a partially defective PLA will be detected, located, and repaired.

CHAPTER 3

Fault-Tolerant Programmable Logic Arrays

Semiconductor device manufacturers continuously strive to increase chip complexity, to reduce the speed-power product, to increase chip reliability, and to produce the most useful and effective devices. As the integrated circuits progress from LSI, VLSI to ULSI (Ultra Large Semiconductor Integration) technology, a single chip may contain 10^6 transistors. The smaller dimension brings greater parasitic capacitance and higher wiring resistance. This leads the slower signal propagation time delay through the necessary wiring when random logic design is used. An example is that the microprocessors lag behind simpler memory chips, if using random logic design [17]. In other words, routing is crucial in ULSI system. On the other hand, complexity index is measured by the regularity factor which is an important role in accomplishing the ULSI. If all the circuits were realized by regular structures, the regularity factor will be high. Regular structures, such as ROM, RAM, PLA, etc., in ULSI design, will be used instead of random logic structures [45] which need more chip area and introduce long propagation time delay due to the necessary routing and placement.

The only integrated circuits so far to have exploited fault-tolerant techniques commercially have been memory chips. With redundancy, partially defective memory chips can be repaired. There are many similarities between memory chips and PLAs that, at first sight, suggest the application of the same redundancy techniques to both these devices for fault-tolerant capability: in a Field PLA (FPLA), for example, a

fault in a product term can result in the term being both logically and physically deleted [49]. This term can be replaced by a fault-free product term supplied by the spares in a manner similar to replacement of rows and columns in a redundant memory.

However, there exist unique conditions in the internal structure of a PLA that severely limit the direct application of memory redundancy techniques. For example, consider those faults, such as *stuck-at* faults in the OR plane, that require spare output lines for repair. The obvious solution is to provide more lines than originally required. In this case, at least in principle, some spare lines in the redundancy design may be reserved to repair this type of fault. In practice however, major routing problems are encountered when the switching of a faulty output signal line to a spare has been accomplished.

In this chapter, a fault-tolerant design of an alternative regular structure, PLA, is presented. A fault-tolerant PLA should be designed in such a way that it is fault diagnosable and repairable during the manufacturing process, and testable in field use. This chapter is organized as follows. The basic structure of a PLA and its fault models are introduced in the first section. In Section 3.2, a repairable PLA (RPLA) design and its repair rules are presented. Before a defective RPLA can be repaired, the locations of defects must be precisely identified. Therefore, a fault diagnosable PLA design is discussed in Section 3.3. Chapter summary is given in Section 3.4.

3.1 Programmable Logic Arrays

A programmable logic array (PLA) is a two-level AND-OR logic network that implements the combinational circuits. By adding the storage elements such as latches and flip-flops, PLA can also realize sequential circuits. PLAs are often used to implement controller, decoder and other glue logics needed between circuit blocks. A typical large PLA may have as many as 50 inputs, 67 outputs, and 190 product terms [32]. Due to the fact that complex chips (and in particular microprocessors)

can be efficiently implemented when using PLAs, a trend towards manufacturing larger programmable chips is expected.

3.1.1 PLA Structure and Notation

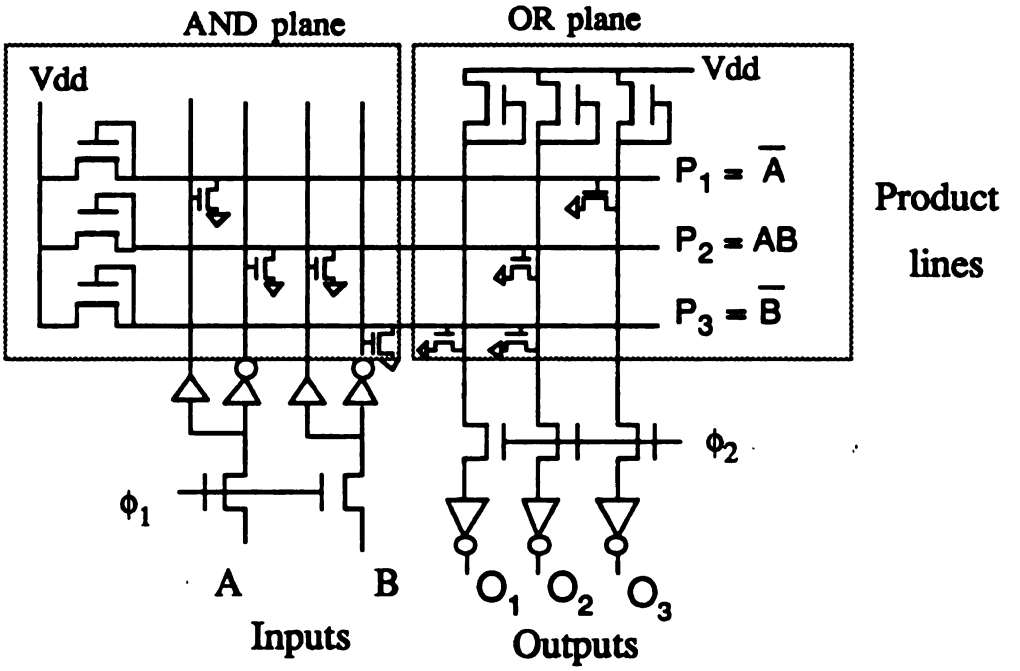
A typical PLA consists of two planes: AND plane and OR plane. Figure 3.1 (a) shows a PLA implemented by a NOR-NOR structure in NMOS technology. The PLA contains two input signals A and B, three output signals O_1, O_2, and O_3 , and three product terms P_1, P_2, and P_3. Figure 3.1 (b) illustrates the logic functions the PLA realized, while Figure 3.1(c) shows the cubic representation of the personality of the PLA, where the cubic notation is listed in Table 3.1.

3.1.2 Fault Models

To design a fault-tolerant PLA, it is necessary to consider the physical defects that are likely to occur in the PLA with the specific technology being used. Three fault models are considered for the NOR-NOR PLA structure in NMOS technology: *crosspoint* faults, *stuck-at* faults, and *bridging* faults [2,41,43,50,54,59].

A *crosspoint* fault is caused by the unintentional presence or absence of a transistor. *Crosspoint* faults can be subdivided into two classes: *missing crosspoint* faults and *extra crosspoint* faults. The former is due to a missing contact at the crosspoint in the AND plane or the OR plane; the latter is due to the unwanted presence of a contact at the crosspoint. The *crosspoint* fault is technological independence.

It is possible to distinguish four types of *crosspoint* faults according to the location of the faults: *growth* faults, *shrinkage* faults, *disappearance* faults, and *appearance* faults. A *growth* fault (or G-fault, for short) is caused by a missing crosspoint in the AND plane, resulting in the disappearance of an input variable from



(a)

Logic function of a PLA	Cubic notation
$O_1 = P_3 = B$	0 - 0 0 1
$O_2 = P_2 + P_3 = AB + \overline{B}$	1 0 0 1 0
$O_3 = P_1 = \overline{A}$	- 1 1 1 0
	A B $O_1O_2O_3$

(b) (c)

Figure 3.1 Programmable Logic Array:
(a) NMOS Implementation; (b) Logic Functions; and
(c) Personality in Cubic Notation.

Table 3.1 Cubic Notation

	AND plane	OR plane
1:	connect to complement input	connect to output
0:	connect to true input	does not connect to output
-:	no connection	not used

a product term (Figure 3.2 (a)). A *shrinkage* fault (or S-fault) is caused by an extra crosspoint in the AND plane, resulting in an additional input variable in a Boolean product term (Figure 3.2 (b)). A *disappearance*, or D-fault, (*appearance*, or A-fault), is due to a *missing* (*extra*) *crosspoint* fault in the OR plane as shown in Figure 3.2 (c) (Figure 3.2 (d)).

Regardless of the technology used in the actual implementation, the logical line *stuck-at* fault model is frequently used. A *stuck-at* fault is the simplest type of fault that can occur in a PLA. A *stuck-at* fault is a line permanently at a logical 1 or 0 state. This can result from the faulty line being opened or shorted to the power line or Ground line (GND). A *stuck-at-0* (s-a-0, for short) fault at an input bit line causes a variable disappearing from an implicant. For instance, as shown in Figure 3.3 (a), the fault-free logic function in the product line was $\overline{A}\,\overline{B}$, and now becomes $\overline{B}$, because the input bit line has an s-a-0 fault. Similarly, an s-a-0 fault at product line changes the function from $\overline{A}B$ to B as shown in Figure 3.3 (b).

A *stuck-at*-1 (s-a-1) faulty input bit line results in s-a-0 faults at those product lines which have contacts in the crosspoints with this faulty line (Figure 3.3 (c)). Similarly, an s-a-1 faulty product line causes s-a-1 faults at those output lines which have contacts with that faulty product line (Figure 3.3 (d)). Finally, the *stuck-at* faults at the output line cause the output lines to permanently be the faulty state.

A *bridging* fault is a short between two adjacent or crossing lines. This fault forces the same logic value to appear in the bridged lines. A *bridging* fault may cause either a logical AND or a logical OR, depending upon the technology being used, of the bridged Boolean functions in the plane of occurrence. In the MOS technology, a wired-AND logic is assumed. *Bridging* faults can occur in both AND and OR planes. It should be noted that a *bridging* fault will cause both true and complement bit lines to have s-a-0 faults if both lines are shorted.

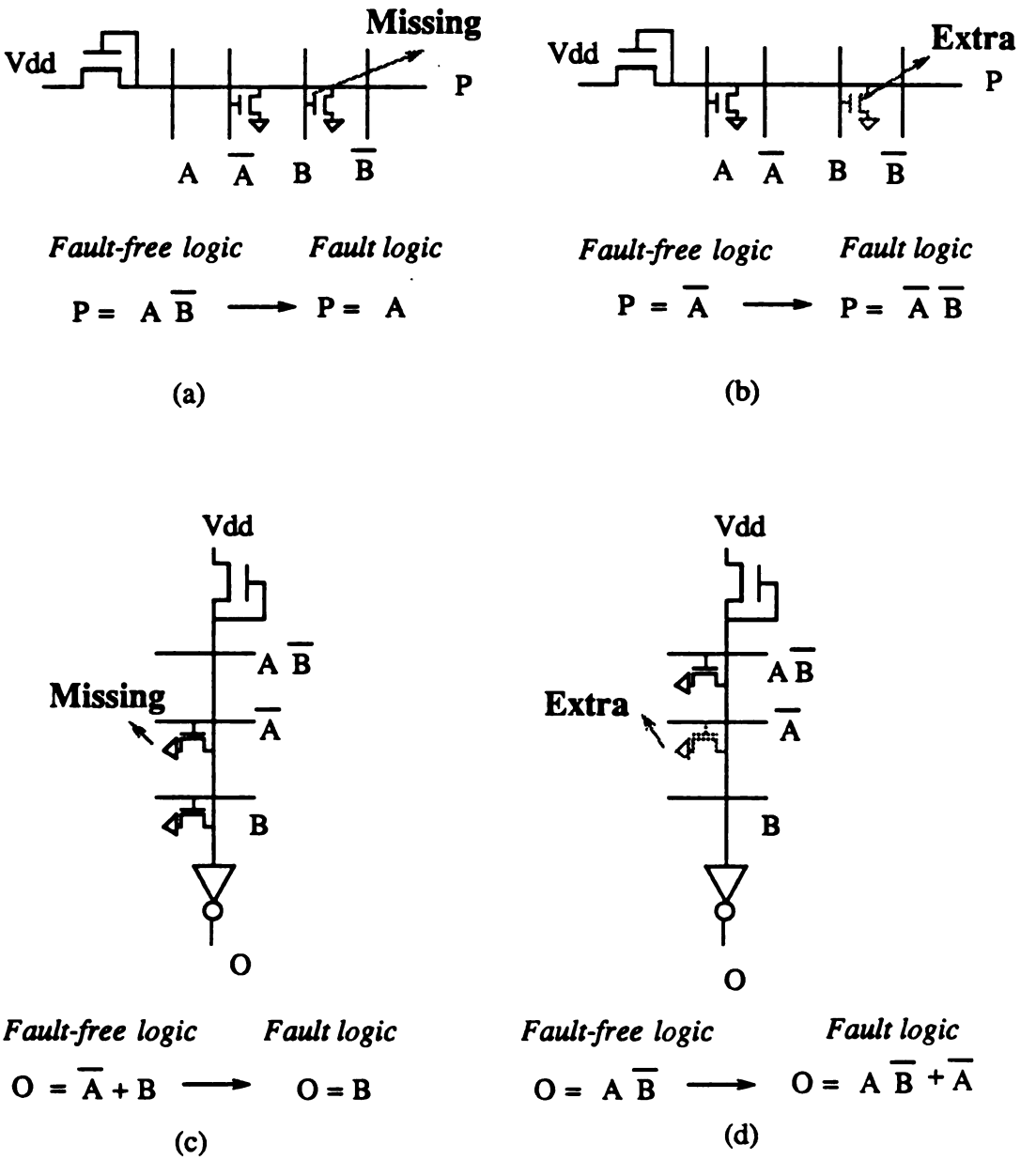


Figure 3.2 Crosspoint Faults:
(a) Growth Fault; (b) Shrinkage Fault;
(c) Disapperance Fault; and (d) Appearance Fault.

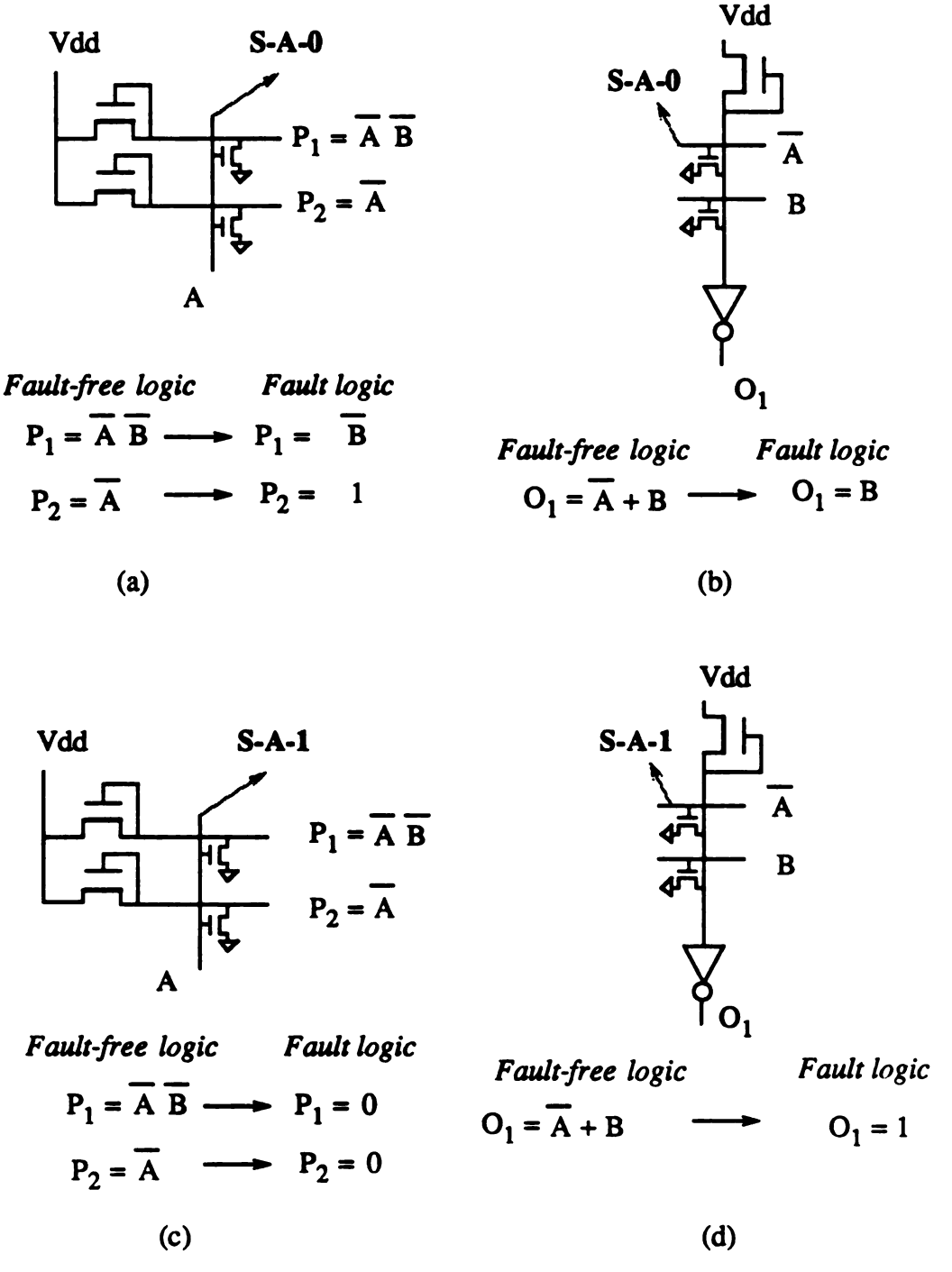


Figure 3.3 Stuck-at Faults:
(a) Input Bit Line Stuck-at-0; (b) Product Line Stuck-at-0;
(c) Input Bit Line Stuck-at-1; and (d) Product Line Stuck-at-1.

3.2 Design of the Repairable PLAs

To avoid complex routing and to repair the faulty PLA, a schematic diagram of a repairable PLA (RPLA) is shown in Figure 3.4. In this design, two spare selector circuits are added internally to control the reconfiguration of the input/output signal lines. The selectors are: the Spare Input Selector Circuit (SISC) and the Spare Output Selector Circuit (SOSC). In addition, several spare lines are also augmented in that design.

To repair a faulty RPLA, a set of repair rules which are based on the fault models discussed in the previous section, must be established. The repair rules are summarized in Table 3.2.

3.2.1 Repair Rules

When a *stuck-at* fault occurs in an input bit line, the line is forced to be either 1 (for s-a-1 fault) or 0 (for s-a-0 fault). A spare bit line programmed with appropriate crosspoints is selected to replace the faulty line, and the faulty line is then disconnected from the SISC circuit shown in Figure 3.4. However, disconnecting the faulty line may cause a "floating" logic. For sake of safety, the disconnected faulty line must be connected to Ground line.

Similarly, a stuck-at faulty output line replaced by a spare output line is disconnected from the SOSC circuit and connected to Ground line. In addition, the faulty output line must be disconnected from the pull-up transistor because a malfunctioning pull-up transistor may cause a short between the power line and the grounded faulty output line.

An s-a-0 faulty product line does not affect the output functions of other product terms of the PLA realized. However, an s-a-1 faulty product line may significantly interfere with the functions, if the faulty line is not repaired. In addition to the use of a spare product line to repair an s-a-1 faulty line, the faulty line must be

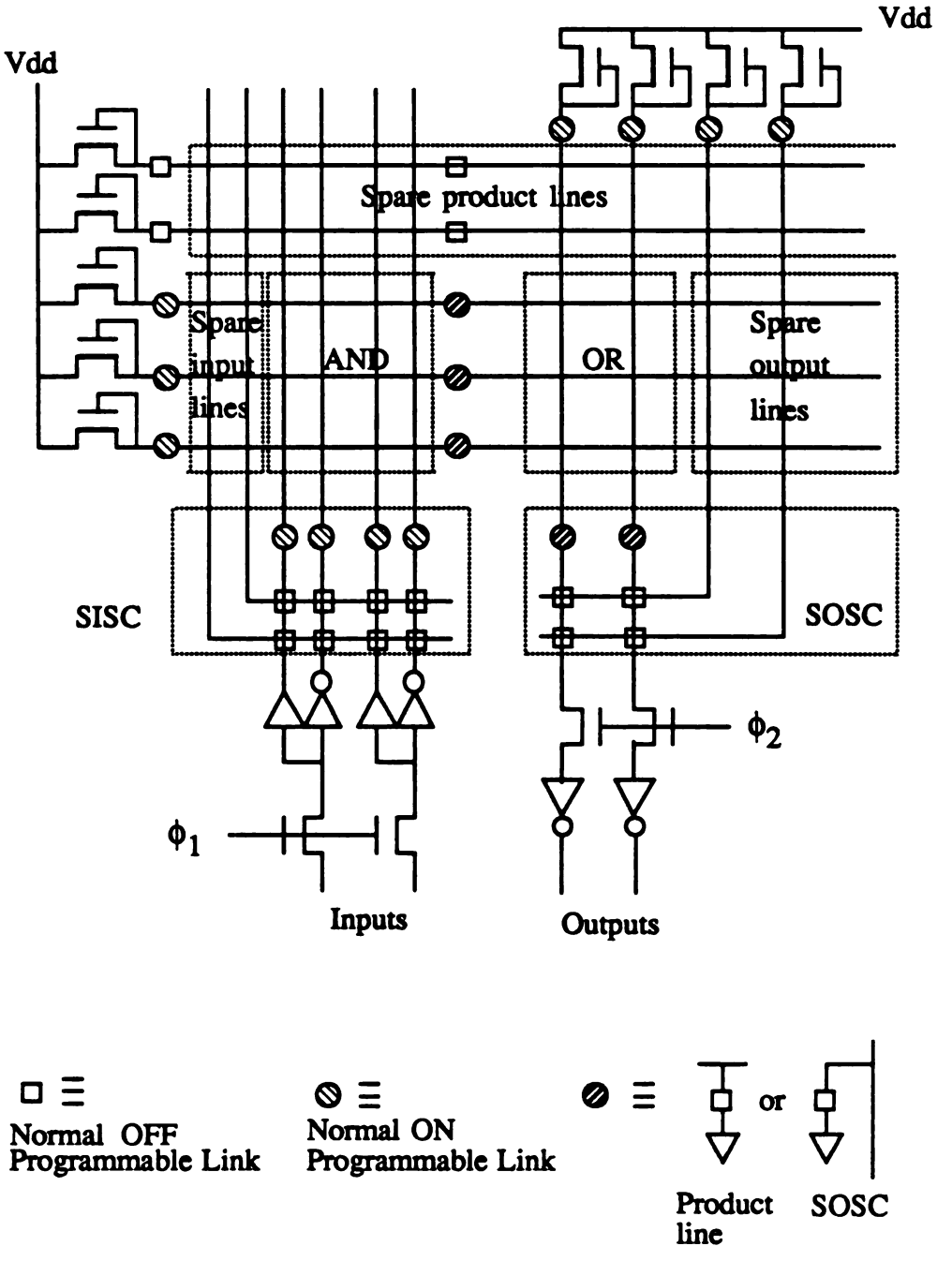


Figure 3.4 Schematic Diagram of a Repairable PLA.

Table 3.2 Repair Rules

Fault Type	Spare line	Faulty Line
Stuck-at fault		
Input bit line	Input bit line	remark 1
Product line	Product line	remark 2
Output line	Output line	remark 3
Crosspoint fault		
Growth	Input bit line	remark 1
	Product line	remark 2
Shrinkage	Input bit line	remark 1
	Product line	don't care
Disappearance	Product line	don't care
	Output line	remark 3
Appearance	Product line	remark 2
	Output line	remark 3
Bridging fault		
Adjacent		
Input bit lines	Input bit lines	remark 1
Product lines	Product lines	remark 2
Output lines	Output lines	remark 3
Crossing		
Input and product lines	Input bit line	remark 1
	and product line	remark 2
Product and output lines	Product line	remark 2
	and output line	remark 3

Remarks:
1. Faulty bit line is disconnected from SISC, and is connected to GND.
2. Faulty product line is disconnected from the pull-up transistor, and connected to GND.
3. Faulty output line is disconnected from the pull-up transistor and from SOSC, and connected to GND.

disconnected from the pull-up transistor and also connected to the Ground line for safety reason.

For repair of the *crosspoint* faults, G- and S- faults are repaired by either spare input lines or spare product lines. Similarly, D- and A- faults are repaired by either spare output lines or spare product lines. In other words, the spare product lines can repair all the four types of *crosspoint* faults. If the *crosspoint* faults are repaired by spare input bit lines or spare output lines, the repair process is the same as the procedure for repairing the *stuck-at* faults. On the other hand, if the *crosspoint* faults are repaired by spare product lines, two cases can be identified: (1) the repair of S- and D- faults; and (2) the repair of G- and A- faults.

An S-fault, with an extra crosspoint in a product line of the AND plane, causes the function realized by the product line to shrink. For example, as shown in Figure 3.5, an S-fault changes the function from A to $A\overline{B}$ due to an extra crosspoint occurring at the true bit line of B. The use of a spare product line programmed with appropriate crosspoints can repair this fault. Since the function realized by the faulty line is included by that of the spare line, the former function is then redundant and does not affect the overall function. Therefore, the faulty line can be retained in the array. However, in order to remove the possible redundancy for high fault coverage, we suggest that the faulty line be disconnected and connected to Ground line. Similarly, D-faults are repaired in the same manner.

Figure 3.6 shows that the function P_{s1} =AB, realized by a spare product line programmed with appropriate crosspoints, is included in P_1= A which is realized by the product line with a G-fault. The use of the spare product line cannot correct the output function as shown. Therefore, the faulty line must be disconnected, i.e., the faulty line is disconnected from the pull-up transistor and connected to the Ground line. Similarly, the A-faults are repaired in the same way.

Bridging faults force the bridged lines to have the same logic. In general, the adjacent *bridging* faults are repaired the same as that of *stuck-at* faults. For the

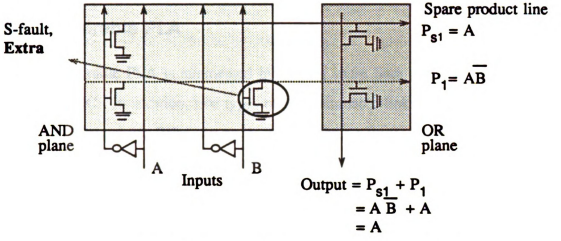


Figure 3.5 The Repair of S-fault with a Spare Product Line.

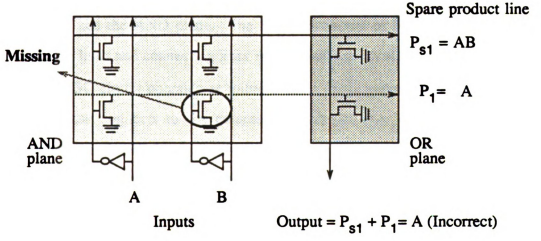


Figure 3.6 The Repair of G-fault with a Spare Product Line.

crossing *bridging* faults, each bridged line is repaired by the same type of spare line as illustrated in Table 3.2.

3.2.2 Repairable PLA

A repairable PLA is augmented by adding spare lines and two control circuits, SISC and SOSC. In addition, two types of programmable links are employed: Normal-on link and Normal-off link. As suggested by their names, the Normal-on (Normal-off) link remains at the ON (OFF) state until the link is programmed; it then alters its state. The programming techniques of Normal-on and Normal-off links have been discussed in Section 2.4 for the designs of fault-tolerant 64K DRAM's [7] and Hi-CMOS 8k × 8 SRAM's [38].

3.2.2.1 SISC and Spare Input Bit Lines

The SISC is added to the input portion of the conventional PLA between the input decoder and the AND plane. The SISC, as shown in Figure 3.7 (a), consists of programmable links and connecting lines with associated circuits.

The SISC circuit operates as follows: prior to the programming of the links, the input signal line connects to the column line through the Normal-on link as in the regular operation of a PLA. Since the Normal-off link is in the OFF state, there is no connection between the input line and the spare input line. When faults are detected and their faulty lines are located, these faulty lines are disconnected from the inputs by opening the Normal-on links. These inputs are then switched to connect spare input lines by programming the Normal-off links to the ON states.

More precisely, the mechanism of the line reconfiguration is described as the switches shown in Figure 3.7 (b), where the S1 witch (equivalent to Normal-on link) is closed and the S2 switches (equivalent to Normal-off links) are opened during the

normal operation. When the faulty input line b_1 is found, for example, suppose that the spare input bit line P_{s1} is assigned to repair it. This can be accomplished as shown in Figure 3.7 (c), where the S1 switch is opened and the S2 switch in the lower connecting line is closed. In this case, the path is formed by connecting the spare input line P_{s1} instead of the faulty input line b_1.

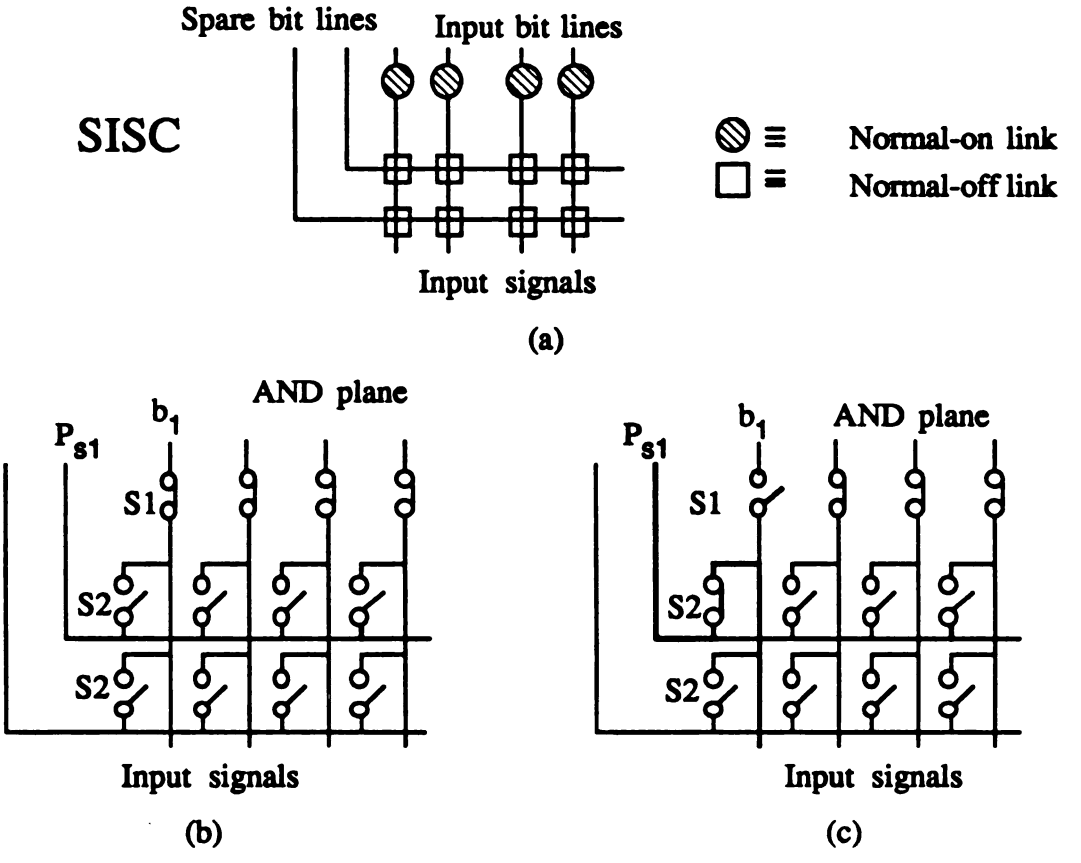


Figure 3.7 Spare Input Selector Circuit (SISC):
(a) Schematic Diagram; (b) Normal Operation; and
(c) Line Reconfiguration.

Figure 3.8 (a) illustrates the stepwise operations of the SISC and spare bit lines. A faulty bit line, as indicated by the dotted line, is repaired by the following steps (Each step is numbered in sequence). First, the faulty line is disconnected

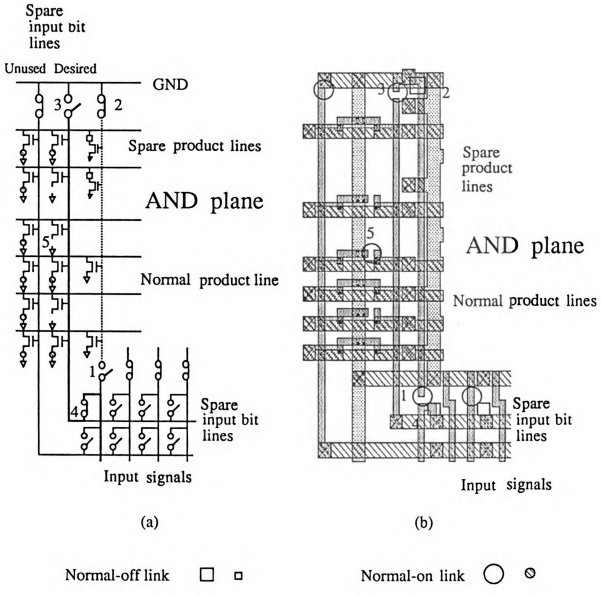


Figure 3.8 The Programming Procedure of SISC and Spare Input Lines: (a) Line Reconfiguration; and (b) Physical Layout.

from the SISC (Step 1), and grounded (Step 2). Then, the desired spare bit line is disconnected from the GND line (Step 3), (for sake of safety, the unused spare bit lines are generally grounded), and connected to input signal through the SISC (Step 4). Finally, the crosspoints in the spare line are programmed (Step 5).

Figure 3.8 (b) shows that the width of a pair of spare input bit lines is 32 λ, and the size of the SISC is $(8s_n + 11) \times 16n \ \lambda^2$, where n and s_n are the numbers of input lines and spare input lines, respectively.

3.2.2.2 SOSC and Spare Output Lines

The SOSC is designed in a fashion similar to the SISC. The output portion of the conventional PLA is modified by inserting the SOSC between the OR plane core and the output inverters.

Similar to the stepwise operations of the SISC, Figure 3.9 (a) shows the programming procedure of the SOSC. First, the faulty line is disconnected from the pull-up transistor (Step 1) and the output line (Step 2), and grounded (Step 3). Then, the desired spare output line is connected to the SOSC (Step 4) and the pull-up transistor (Step 5), (for safety, the unused spare output line is disconnected from the pull-up transistor). Finally, the appropriate crosspoints in the spare output line are programmed.

Figure 3.9 (b) shows that the width of a spare output line is 22 λ, and the length of the SOSC is $(8s_m+14) \lambda$, where s_m is the number of spare output lines and the extra Ground signal line is 14 λ in length.

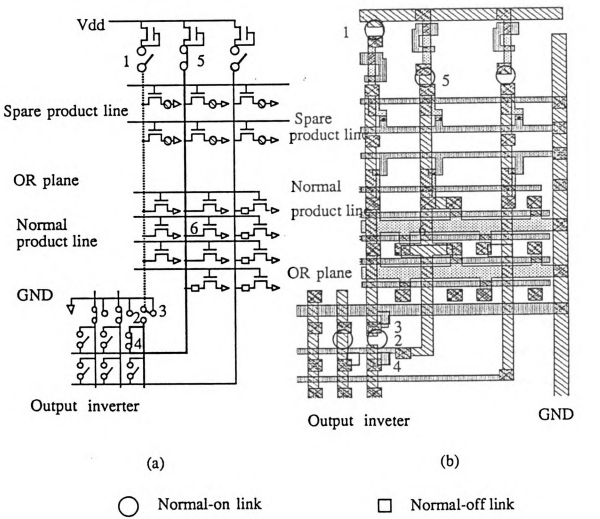


Figure 3.9 The Programming Procedure of SOSC and Spare Output Lines: (a) Line Reconfiguration; and (b) Physical Layout.

3.2.2.3 Spare Product Lines

Figure 3.10 illustrates the stepwise programming procedure of the use of spare product lines. First, the faulty product line is disconnected from the pull-up transistor (Step 1), and grounded (Step 2). Then, the desired spare product line is programmed so that the line is disconnected from the Ground line (Step 3) and connected to the pull-up transistor (Step 4). Finally, the crosspoints in the spare line are programmed (Step 5 and 6).

Figure 3.10 (b) shows that a spare product line is 22 λ in width.

3.2.3 Automatic Layout Generator

The computer-aided design (CAD) tools play very important roles in VLSI design. They can reduce the turnaround time and make design changes more quickly. In this study, an automatic layout generator, *MRPLA* [10], has been developed and implemented in Sun 3/160 for generating the physical layout of the repairable PLA.

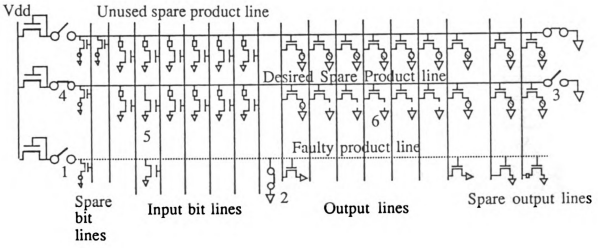
MRPLA requires a template when generating the repairable PLAs. The template contains rectangles, or *tiles*, filled with mask information. These *tiles* are labelled with names that can be called by *Mquilt* [36] routine to be aligned according to a certain semi-regular structure which *MRPLA* will define.

The first step in making a *MRPLA* template is to design a sample RPLA, as shown in Figure 3.11. This RPLA should include at least one example of each possible combination of template cells. With this template, *MRPLA* will have a correct example to generate its own larger RPLAs.

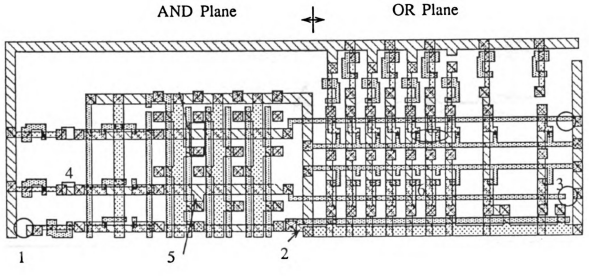
Once a template has been designed for a sample RPLA, *tiles* can be defined for each cell in the template. There are 12 groups of *tiles* used in the *MRPLA* template. They are

(1) the core of the AND plane;

(2) the core of the OR plane;



(a)



(b)

○ Normal-on link □ Normal-off link

Figure 3.10 The Programming Procedure of Spare Product Lines:
(a) Line Reconfiguration, and (b) Physical Layout.

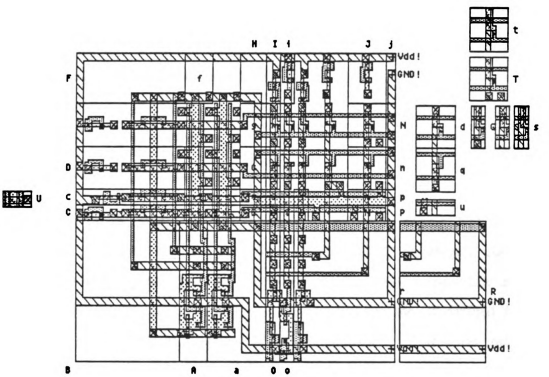


Figure 3.11 A Sample RPLA Template.

(3) the sides of the AND plane;

(4) the sides of the OR plane;

(5) the top and bottom of the AND plane;

(6) the top and bottom of the OR plane;

(7) the *tiles* between planes;

(8) the *tiles* of the spare input lines;

(9) the *tiles* of the spare output lines;

(10) the *tiles* of the SISC;

(11) the *tiles* of the SOSC;

(12) the *tiles* of the spare product lines.

Each *tile* in the sample RPLA template, as shown in Figure 3.11, is labelled by a character. An array with the predefined characters will represent a user-defined RPLA. For example, consider an RPLA whose personality and spare line allocation are specified in Figure 3.12 (a) and (b). According to the predefined labels, a character array, as shown in Figure 3.12 (c), is generated to describe this RPLA. Finally, the physical layout of this RPLA is generated by *MRPLA* and shown in Figure 3.12 (d).

The *MRPLA* is currently developed for the RPLA design in NMOS technology. However, extension to other technology can be readily made. The automatic layout generator *MRPLA* is more feasible than generating layout by using a graphic editor.

3.2.4 Performance

According to the physical layout generated by *MRPLA*, both chip area and propagation delay time are evaluated.

```
10 01
0- 10
-1 11
```

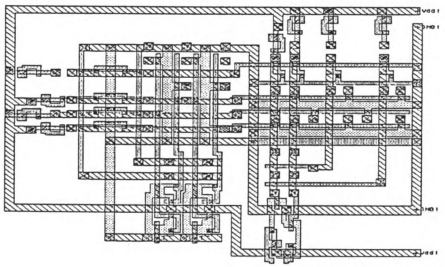
(a)

```
2 inputs,
3 product lines,
2 outputs,
2 spare input lines,
1 spare product line,
2 spare output lines
```

(b)

```
FffHIiJJj
DbbgkkLLN
c1Xe24mmp
CXxE36MMP
cx1e44mmp
BAaOoR
```

(c)



(d)

Figure 3.12 *MRPLA*: (a) PLA Personality, (b) Spare Line Allocation, (c) Character Array, and (d) Physical Layout Generated.

3.2.4.1 Chip Area

To assess the chip area requirement of the designed RPLA, a floor plan is shown in Figure 3.13. All dimensions are given in units of lambda. The floor plan is sketched according to the actual dimension of the physical layout generated by *MRPLA*.

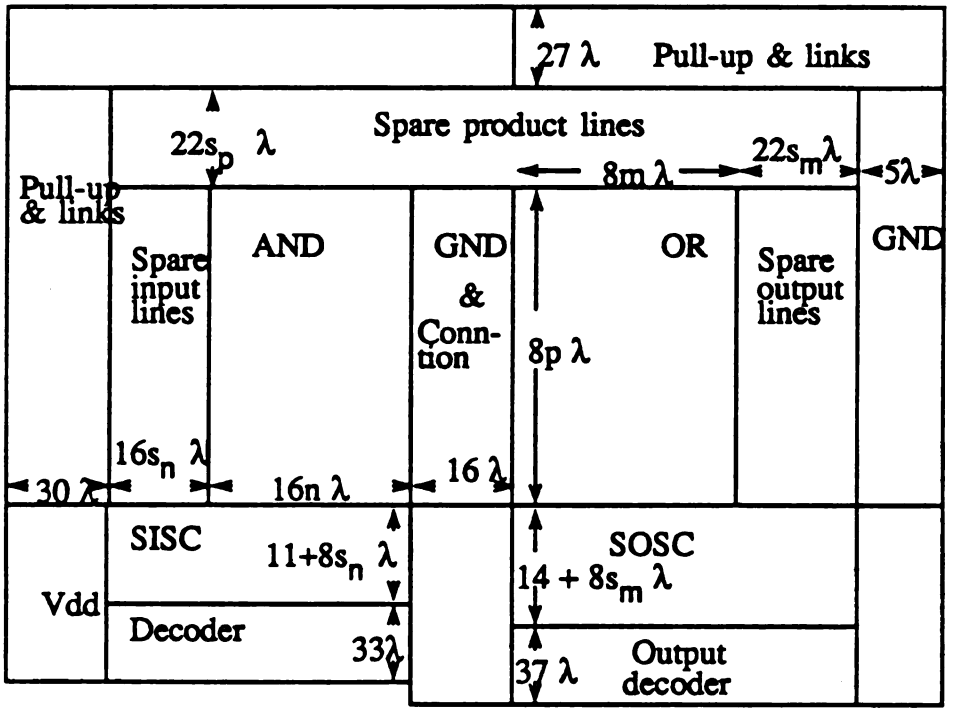


Figure 3.13 Floor Plan of an (s_n,s_p,s_m)-RPLA.

For simplicity, let an (s_n,s_p,s_m)-RPLA be denoted as an RPLA with s_n spare input bit lines, s_p spare output lines, and s_m spare product lines. Table 3.3 lists the area overhead for various spare line assignments, where the area of an (s_n,s_p,s_m)-RPLA is estimated as:

$$\text{Area of an } (s_n,s_p,s_m)\text{-RPLA} = (30 + 16(n + s_n)) \times (71 + 8p + 8s_n + 22s_p) + (21 + 8m + 22s_m) \times (78 + 8p + 22s_p + 8s_m)$$

Following the physical layout generated by the UCB tool *MPLA* [40], a floor plan for a conventional PLA is shown in Figure 3.14. Thus, the area is estimated as:

Original PLA area = $64(2n + m)p + 8(152n + 76m + 49p) + 3724$

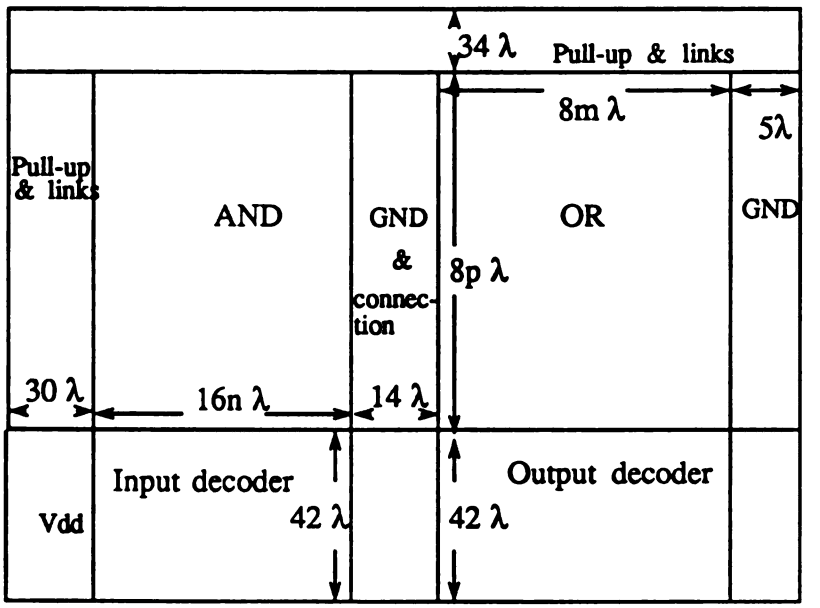


Figure 3.14 Floor Plan of the PLA.

Table 3.3 Area Overhead in RPLAs

			Original PLA	(1,2,1) - RPLA		(2,2,2) - RPLA		(2,3,2) - RPLA	
n	p	m	Area	Area	%	Area	%	Area	%
50	190	67	2210460	134868	6.10%	209160	9.46%	241346	10.92%
60	200	60	2495564	142564	5.71%	220728	8.84%	255202	10.23%
100	200	100	4104524	189924	4.63%	275768	6.72%	331362	8.07%
100	400	100	8022924	253924	3.16%	400568	4.99%	456162	5.69%

3.2.4.2 Propagation Delay Time

The propagation delay time is based on the assumption that the delay time of a logic gate is directly related to the driving capability of its transistor. While the pull-up time is limited by the effective load capacitance and the charging current provided by the pull-up transistor, the pull-down time is determined by the effective load capacitance and the charging current drained by the pull-down transistor. The pull-up time is usually longer than the pull-down time in NMOS technology and is commonly considered as the delay.

An over-simplified delay time model could be written as $T = \kappa C_{load}$, where κ is a constant determined by the average charging time and the high state output voltage, and C_{load} is the effective load capacitance [37]. Eventually, the effective load capacitance is contributed by the transistor gate capacitance, C_g, and signal path capacitance, C_p. The transistor gate capacitance is due to the oxide interposed between the gate and the substrate of a pull-down transistor. The signal path capacitance is defined as the capacitance presented by a signal path of 8 λ that is approximately equal to the spacing between two product lines, or half of the spacing between two input lines.

The delay time penalty is defined as the increased delay time for the redundancy delay. The delay time penalty for the AND plane of the RPLA in Figure 3.15 (a) is $2n / (2n+15.5p)$ [64]. Similarly, the delay time penalty for the OR plane of the RPLA is $m/(m+15.5p)$. The delay time penalty for the RPLA is $2n/(2n+15.5p) + m/(m+15.5p)$. The detailed derivation of the above delay time penalty can be found in Appendix 1. Table 3.4 shows the delay time penalty for various sizes of RPLAs.† It should be noted that the delay penalty can be alleviated by the spare line allocation shown in Figure 3.15 (b) [64].

Table 3.4 Delay Time Penalty of the RPLAs

n	p	m	Delay Time Penalty
50	190	67	5.5%
60	200	60	5.6%
100	200	100	9.2%
100	400	100	4.7%

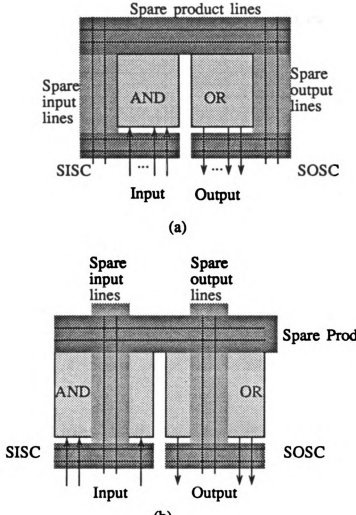


Figure 3.15 Different Allocation Schemes of Spare Lines:
(a) Spares Placed in the Border; and (b) in the Middle.

3.3 Design of the Diagnosable PLA

The design of RPLAs has shown that partially defective chips can be repaired without reconfiguring the external routing [64,65]. The design has enhanced the chip yield significantly [62]. However, before a defective PLA can be repaired, the location of the defects must be precisely identified.

In this section, the design of a fault-diagnosable PLA (FDPLA) is presented to achieve full diagnosability of single and multiple *stuck-at* faults, *bridging* faults, and *crosspoint* faults. The design of an FDPLA requires that the design must be capable of detecting, locating, and repairing faults during the manufacturing process, and also capable of performing chip testing in field use.

3.3.1 Augmented Circuits

To fulfill the above design requirements, a schematic diagram of a fault diagnosable PLA (FDPLA) is presented as shown in Figure 3.16. The original PLA is augmented by control circuits and two shift registers: the input lines' shift register (ISR) and the product lines' shift register (PSR). In addition to the scan signals (S_{in} and S_{out}) and the non-overlapped clock signals (ϕ_1 and ϕ_2), five extra control signals (M_P, M_I, R, W, and Vdd1) are needed to operate the ISR and PSR.

3.3.1.1 Product Lines' Shift Register (PSR)

Figure 3.17 shows the schematic diagram of a PSR cell. The number of PSR cells in the proposed FDPLA is half of the number of product lines, i.e., each PSR cell's output (labeled as "Next Value") is shared by two adjacent product lines through multiplexing. Multiplexing is used because no more than one shift register cell can fit into the narrow 16 λ width shared by two product lines. The PSR is used

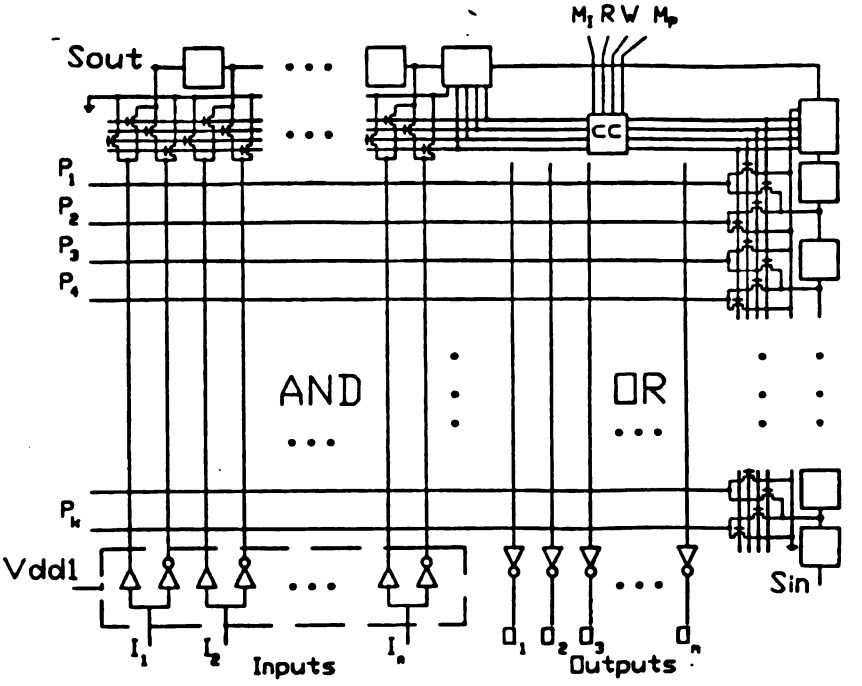


Figure 3.16 A Schematic Diagram of a Fault-Diagnosable PLA.

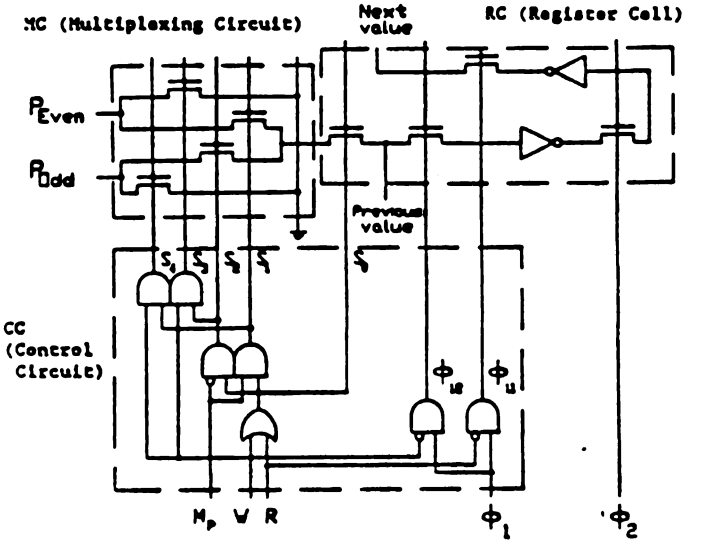


Figure 3.17 A Product Lines' Shift Register (PSR) Cell.

to disable all product lines but one. This allows the effect of a single product line on the output to be observed.

Consider the shift register cell commonly used for testable PLA design, as shown in Figure 3.18. When the cell holds a logic 1, the pass transistor is conducting and therefore the product line is disabled (forced to 0). Conversely, when the cell holds a logic 0, the product line is enabled. As a result, the shift register cell allows the product lines to be disabled. This operation is referred to as the "write" operation.

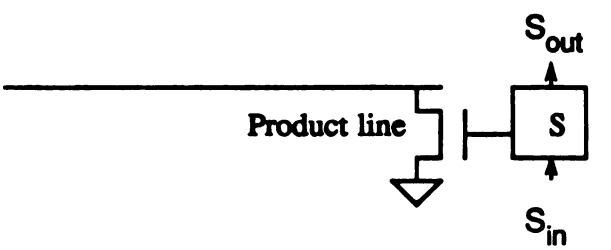


Figure 3.18 The Function of Shift Register Cell in Testable PLA Design.

In contrast, the proposed PSR possesses both "read" and "write" functions. A PSR cell is connected with a control circuit (CC) that is used to control all register cells of the PSR. For simplicity of discussion, let the internal signal lines be labeled by S_0-S_4, and the external signals be denoted as M_p (multiplexing), R (read), and W (write). The signal S_0 (= R+W, where "+" is an "OR" function) is used to control the pass transistor which connects the register cell (RC) and the multiplexing circuit (MC). When S_0=0, the RC is disconnected from the MC. However, when S_0=1, i.e., either read or write (but not both), the data transmission is enabled. Table 3.5 illustrates both the internal signals generated by the CC and the corresponding operations performed for the combinations of external input signals.

It should be noted that, if only the i-th cell of the PSR holds a logic 1, then the assignment (M_p,R,W)=(0,0,1) writes 1 to the (2i-1)-th product line (or P_{2i-1}) and all 0's to the remaining lines, i.e., the assignment only enables P_{2i-1} but disables the remaining product lines. In summary, the proposed PSR not only can read the contents of the product lines, but also can enable one product line at a time to enhance the testability and diagnosability.

Table 3.5 Operations of the PSR

W	R	M_p	S_1	S_2	S_3	S_4	Operations
0	0	x	0	0	0	0	Isolate PSR from the PLA
0	1	0	0	1	0	0	Read ODD numbered product line
0	1	1	1	0	0	0	Read EVEN numbered product line
1	0	0	0	1	1	0	Write data of RC to ODD and set EVEN to 0
1	0	1	1	0	0	1	Write data of RC to EVEN and set ODD to 0
1	1	x	x	x	x	x	Invalid Case (R×W≠1)

Remark: "ODD" ("EVEN") -- odd (even) numbered product line.
R - read; W - write for PSR.

3.3.1.2 Input Lines' Shift Register (ISR)

Figure 3.19 shows an ISR cell, where the number of the ISR cells is the same as that of the inputs. The ISR is operated in a similar fashion as the PSR. The ISR allows reading the contents of the input bit lines and writing the data held in the RC to the bit lines. In order to reduce extra external control signals, the read and write operations in both the ISR and PSR are arranged in opposite ways, i.e., when ISR writes data to the input lines, the PSR reads the contents of the product lines, and vice versa. Specifically, R=1 sets the PSR to the "read" mode and the ISR to the "write" mode, while W=1 sets the PSR to the "write" mode and the ISR to the "read" mode. The control circuit (CC) generates the internal signals S_5-S_8 for the MC of the ISR. Similar to Table 3.5, Table 3.6 illustrates both the internal signals generated by

the CC and the corresponding operations performed for the combinations of external inputs.

Similarly, the ISR can read the contents of each bit line, and also can enable one bit line at a time to improve the testability and diagnosability.

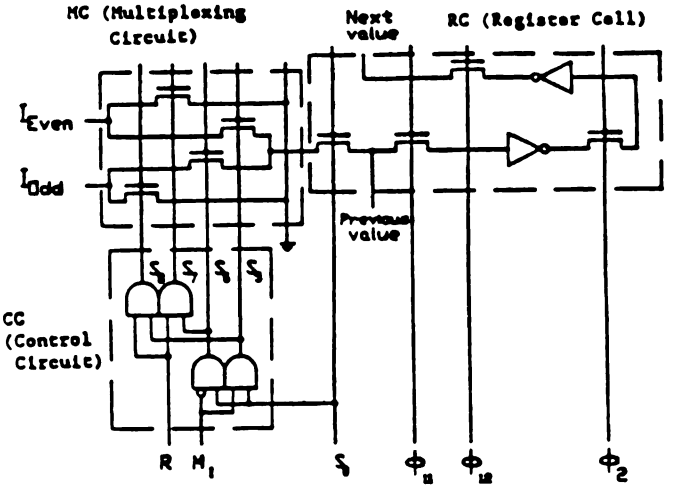


Figure 3.19 An Input Lines' Shift Register (ISR) Cell.

Table 3.6 Operations of the ISR.

W	R	M_I	S_5	S_6	S_7	S_8	Operations
0	0	x	0	0	0	0	Isolate ISR from the PLA
0	1	0	0	1	0	0	Read COMP (complemented) bit line
0	1	1	1	0	0	0	Read TRUE (true) bit line
1	0	0	0	1	1	0	Write data of RC to COMP and set TRUE to 0
1	0	1	1	0	0	1	Write data of RC to TRUE and set COMP to 0
1	1	x	x	x	x	x	Invalid Case (R×W≠1)

Remarks: "ODD" ("EVEN") -- complemented (true) bit line.
R - write; and W - read for ISR.

3.3.1.3 Extra Power Line Vdd1

In order to allow patterns to be applied to input bit lines either through the input lines I_j's or the ISR cells, an extra power line Vdd1 for the input decoder is used. If the patterns are applied through I_j's, the Vdd1 is set to a logic 1; otherwise, a logic 0 is set.

3.3.2 Design Evaluation

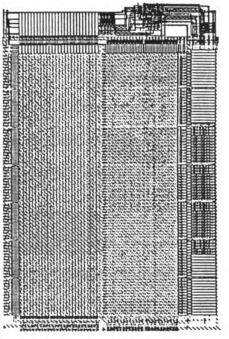
Figure 3.20 (a) shows the physical layout of a FDPLA that includes the original PLA, the spare lines and control circuits for repair, and the added shift registers for fault diagnosis. According to the floor plan of Figure 3.20 (b), each register cell layout of the ISR and PSR takes $16 \times 140\ \lambda^2$ in area, i.e., the augmented area for FDPLA is:

$$\text{Augmented area} = (2n + p) \times 8 \times 140$$

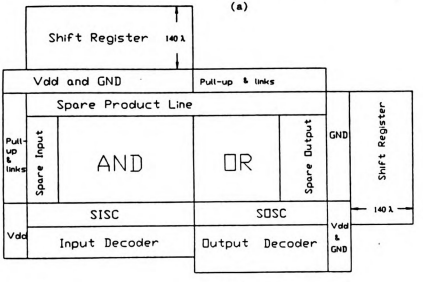
Table 3.7 lists the area overhead for the original PLA, FDPLA, and (1,2,1)-RPLA.

Table 3.7 Area Overhead of FTPLAs

			Original	Augmented PLA				
			PLA	(1,2,1)-RPLA		FDPLA		FTPLA
n	p	m	Area	Area	%	Area	%	%
50	190	67	2210460	134868	6.10%	324800	14.69%	20.80%
60	200	60	2495564	142564	5.71%	358400	14.36%	20.07%
100	200	100	4104524	189924	4.63%	448000	10.91%	15.54%
100	400	100	8022924	253924	3.16%	672000	8.38%	11.54%



(a)



(b)

Figure 3.20 Fault-Diagnosable PLA:
(a) Physical Layout; and (b) Floor Plan.

3.4 Summary

Low yield problem usually happens in a newly developed pilot technology. In order to ensure that large chips are manufactured with a reasonable yield level, a design of repairable programmable logic arrays (RPLAs) has been proposed, in which the partially defective chips can be repaired without reconfiguring the external routing. However, before a defective RPLA can be repaired in the manufacturing process, the locations of the defects must be precisely identified and spare lines have to be optimally allocated. After packaging, it is desirable that the RPLA chip is easily testable in field use.

In this chapter, a fault-tolerant PLA design has been presented. The design achieves full diagnosability of single and multiple *stuck-at* faults, *bridging* faults, and *crosspoint* faults. The detailed diagnosis process will be discussed in the next chapter.

Although the proposed FDPLA design achieves full diagnosability it requires 7 extra signals (W, R, M_p, M_l, Vdd1, S_{in}, and S_{out}) and two sets of shift registers (ISR and PSR). Extra signals imply the increase of pin overhead. In our implementation, however, the signals, such as R, M_l, Vdd1, and S_{out}, are used only for fault location purpose. Therefore, the signals can be applied or measured by using the internal pads of the package. In other words, the signals will not cause any pin overhead. Furthermore, as shown in Figure 3.21, the signals W, M_p, and S_{in} are, in fact, the only pin overhead which is common to the easily testable PLA design [25,54].

The results of this study show that the proposed ISR not only can read the contents of the bit lines, but also can enable the bit lines, one at a time. Due to the "read" and "write" functions, the proposed shift register is better than the conventional shift register in its function. In addition, the basic cell's layout of the proposed shift register is 16 λ wide and 140 λ long. The proposed shift register cell is

smaller than that of the testable PLA design in [59], 16 x 170 λ^2, by nearly 18% in area.

Although the FTPLA design requires 10% to 25% overhead in chip area, the salient features of detecting, locating and repairing faults in this design have demonstrated its feasibility.

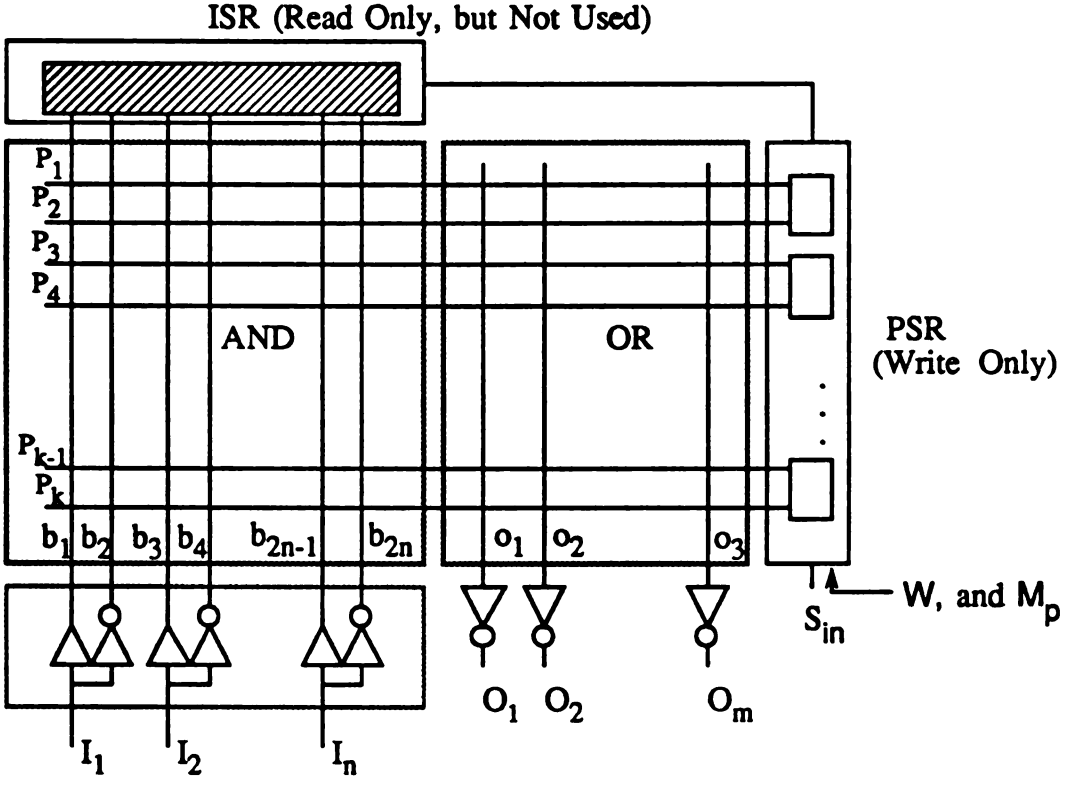


Figure 3.21 An Easily Testable PLA Modified from the FDPLA.

CHAPTER 4

Fault Diagnosis and Repair Process

As mentioned previously, the design requirements of a fault-tolerant PLA are: (1) the PLA should be able to detect, locate, and repair faults during the manufacturing process; and (2) the PLA must be easily testable when the packaged PLA is used in field. To achieve these goals, a fault diagnosable PLA design and its modification for an easily testable PLA design have been presented in the previous chapter.

This chapter describes the fault diagnosis and repair process for the fault diagnosable PLA design. Two examples will be given to demonstrate that the proposed fault diagnosable PLA achieves a full diagnosability of single and multiple *stuck-at*, *bridging*, and *crosspoint* faults. This chapter also presents a simple test process for detecting faults in a fault-tolerant PLA that is packaged and used in field.

4.1 Locate and Repair Faults in Manufacturing Process

In this section, a fault location and repair process is presented. The faults include single and multiple *stuck-at*, *bridging*, and *crosspoint* faults. The proposed process consists of four major steps: (1) detect faults in augmented circuits; (2) identify and repair faults in the AND plane; (3) identify and repair faults in the OR plane; and (4) repair *crosspoint* faults.

More specifically, we first test the augmented circuits. Augmented circuits are non-redundant; any faults in the added circuits are considered as fatal, and the repair of the PLA is unnecessary. Once the augmented circuits have functioned properly, we locate and repair faults in both planes. To identify faults in the AND plane, we set the ISR to the "read" mode to observe the status of the input bit lines to locate both *stuck-at* and *bridging* faults. This is followed by setting the ISR to the "write" mode and the PSR to the "read" mode for reading the contents of the product lines. Then we can locate the *stuck-at* and *bridging* faults at the product lines, and G- and S-faults.

It should be noted that, in order to precisely locate faults, both *stuck-at* and *bridging* faults must be repaired immediately when they are identified. Otherwise, a *stuck-at*-1 faulty bit line, for example, will cause those product lines which have contacts in the crosspoints to have *stuck-at*-0 faults. This would produce some difficulties in precisely locating the faults and identifying the fault types. Once these faults have been repaired, the remaining *crosspoint* faults are repaired by efficiently utilizing the spare lines.

Finally, faults in the OR plane are identified by setting the PSR to the "write" mode and observing the output lines. We can also locate both *stuck-at* and *bridging* faults at the output lines, and D- and A- faults. After the *stuck-at* and *bridging* faults are repaired, a spare allocation algorithm is applied to efficiently repair the *crosspoint* faults.

4.1.1 Detect Faults in Augmented Circuits

The shift register chain used in the proposed FDPLA includes the ISR, PSR, and some extra register cells for observing the control signals. To test the shift register chain, we first isolate the shift register cells from the multiplexing circuits, as shown in Figures 3.18 and 3.19, by setting both signals R and W to logic 0's, i.e., $S_0 = 0$. Then, we apply a scan pattern (0101..01) to the shift registers to detect the

stuck-at faults. Since the signals generated by the control circuits can be observed from the additional shift register cells, the control circuits are also fully testable. In this step, any fault is considered fatal and no further testing is needed for the unrepairable PLA.

Once no fault has been detected in the augmented circuits, the following fault location procedure for both planes is utilized. A simplified diagram for the fault diagnosable PLA, as shown in Figure 4.1, is used to describe the stepwise fault location procedure.

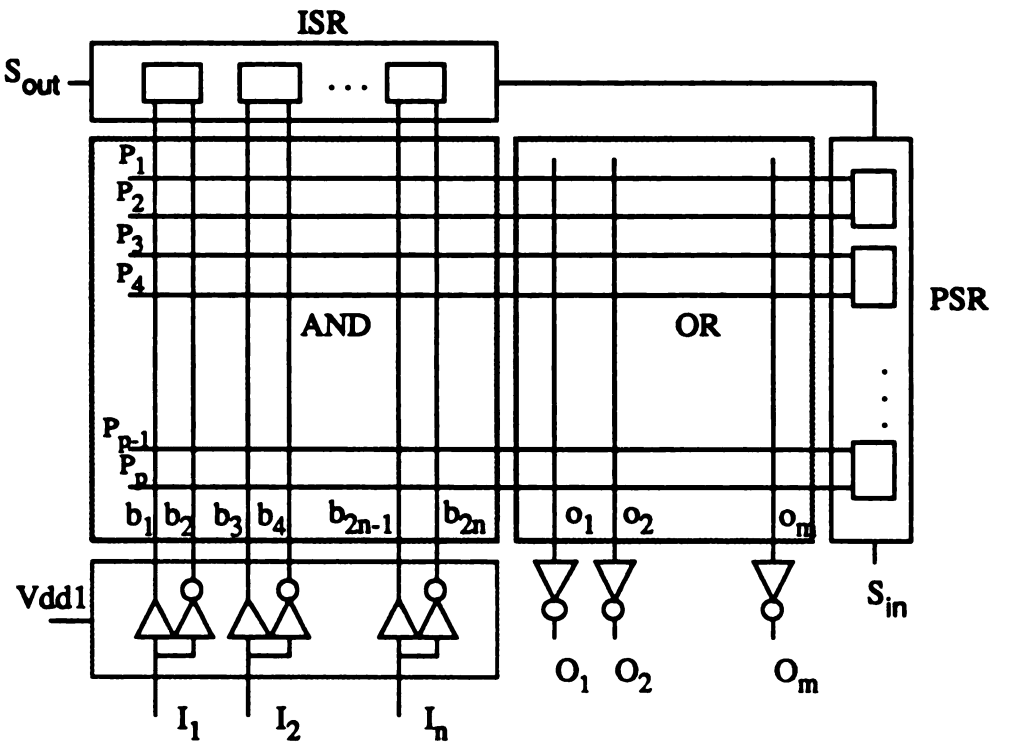


Figure 4.1 A Simplified Diagram for Fault-Diagnosable PLA.

4.1.2 Identify and Repair Faults in the AND Plane

The faults in the AND plane include: (1) *stuck-at* faults at input bit lines, (2) *bridging* faults at adjacent input bit lines, (3) *bridging* faults between input bit lines and product lines, (4) *stuck-at* faults at product lines, and (5) G- and S- faults. The *bridging* faults at adjacent product lines are considered in the next step.

Both *stuck-at* and *bridging* faults at the input bit lines can be easily tested and located if the contents of the bit lines can be observed. Specifically, since both true and complemented bit lines are arranged in sequence, the contents of the bit lines are expected to be EP1=(1,0,1,0,...,1,0) for the applied input pattern $(I_1,...,I_n)=(1,...,1)$, and EP0=(0,1,0,1,...,0,1) for the pattern (0,0,...,0), i.e., EP0 is the bitwise complement of EP1. As such, a *stuck-at* fault is located at the i-th bit line if $EP0_i=EP1_i$.

Figure 4.2 illustrates the applied patterns for locating *stuck-at* faults and *bridging* faults. We first apply the test patterns through I_1-I_n by setting Vdd1 = "1"; set the ISR to the "read" mode by assigning (R,W)=(0,1); and write 0's to all product lines by shifting all 0's to the PSR, and by assigning M_p=0. (It should be noted that if the product line has an s-a-1 fault, the product line will never be set to the 0 state.) Consequently, the contents of bit lines can be read from the ISR, where assigning M_I=1 (0) reads the contents of the true (complemented) bit lines. The contents are loaded to EP0 or EP1 for the applied 0's or 1's input patterns, respectively.

Property 1.

If one of the bridged lines has a *stuck-at* fault, the *bridging* faults are equivalent to *stuck-at* faults.

Proof. Since bridged lines should have the same logic, a bridged line with a *stuck-at* fault will force the remaining lines to have the same *stuck-at* fault. □

For the case that none of the bridged lines has a *stuck-at* fault, the following five cases can be identified.

(1) input bit line adjacent *bridging* fault (Type BRII fault),

(2) product line adjacent *bridging* fault (Type BRPP fault),

(3) output line adjacent *bridging* fault (Type BROO fault),

(4) one input bit line bridges to a product line (Type BRIP fault), and

(5) one product line bridges to an output line (Type BRPO fault).

Property 2.

Types BRII and BRIP faults are equivalent to s-a-0 faults.

Proof. For the type BRII, two cases can be identified here. The first case is the *bridging* faults between the true bit line and the complemented bit line of the same input j. The second case is the short between the complemented bit line of input j and the true bit line of input j+1. In either case, because of no *stuck-at* faults at the bridged lines and the wired-AND *bridging* fault model, the bridged bit lines are diagnosed as having s-a-0 faults. For the type BRIP, since the bridged product lines do not have *stuck-at* faults, the assigned 0-value product lines can force the bridged bit line to be s-a-0, i.e., the BRIP *bridging* faults at the bridged bit lines are equivalent to s-a-0 faults. □

Both Properties 1 and 2 have shown that the *bridging* faults are equivalent to *stuck-at* faults, and thus they have been located and repaired as *stuck-at* faults. As mentioned previously, the *stuck-at* faults must be repaired immediately once they are located. After the faults are repaired by the spare lines, the above procedure is carried out again to assure that neither *stuck-at* faults nor *bridging* faults occur at the assigned spare lines.

Figure 4.3 illustrates the process for locating *stuck-at* faults and *bridging* faults at the product lines, as well as G- and S- faults. These faults can be identified if we observe the contents of the product lines. More specifically, we first apply test

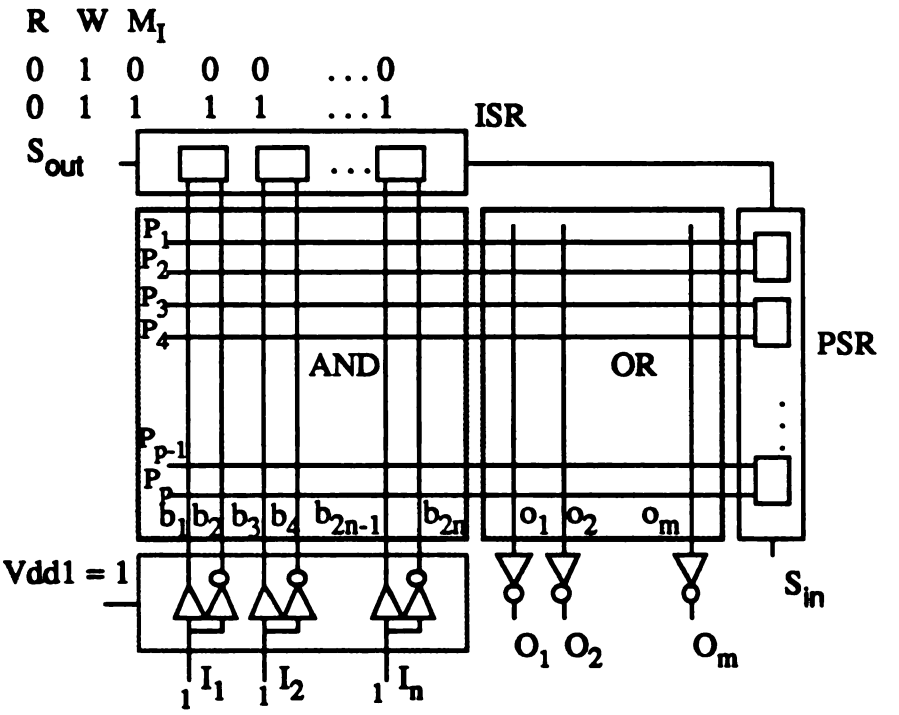


Figure 4.2 Identify Input Line Stuck-at Faults and Bridging Faults.

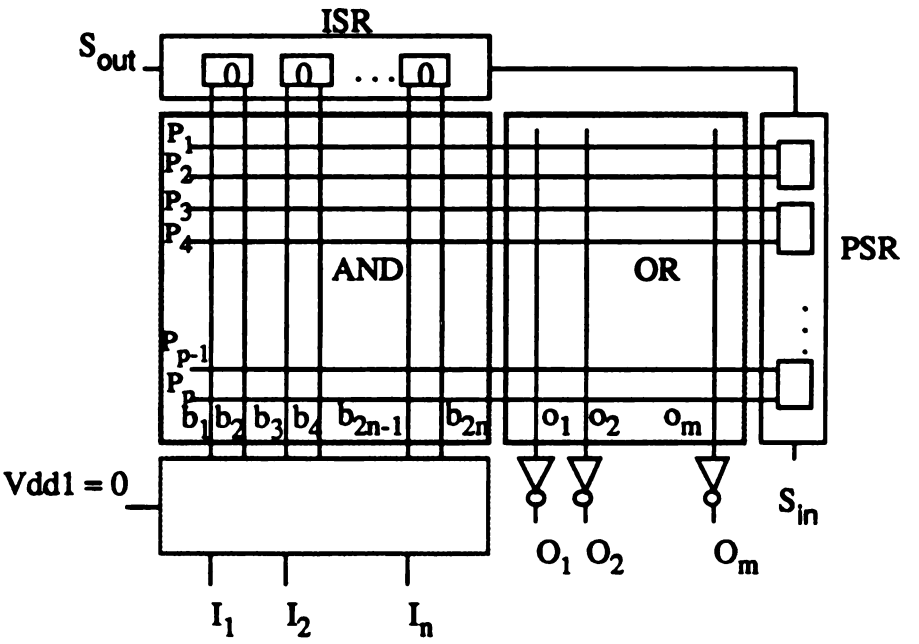


Figure 4.3 Identify Faults at Product Lines as well as G- and S- Faults.

patterns through the ISR by assigning Vdd1 = "0," set the ISR to the "write" mode and the PSR to the "read" mode by assigning (R,W)=(1,0); and write all 0's to the ISR with M_I=0 (and change to M_I=1 later). Then, we expect to have all 1's in the PSR. If an unexpected result occurs at the j-th product line, this implies that the line has an s-a-0 fault.

After the s-a-0 faults have been located and repaired, a walking "1" pattern is applied to the ISR. When only the j-th cell of the ISR holds 1, M_I=1 results in a logic 1 in the true bit line (b_{2j-1}), but all 0's in the remaining bit lines, i.e., this assignment enables only the bit line b_{2j-1} and disables all other bit lines, and allows us to observe one bit line at a time. The status of the crosspoints in each enabled bit line are recorded and form the following matrix.

$$AP_F = \begin{bmatrix} A_{11}\, A_{12} \cdots A_{1r} \\ A_{21}\, A_{22} \cdots A_{2r} \\ \cdots\cdots \\ \cdots\cdots \\ A_{p1}\, A_{p2} \cdots A_{pr} \end{bmatrix} \qquad (4.1)$$

where A_{ij} = 0 (1) represents the absence (presence) contact of the crosspoint. It should be noted that the statuses were observed from the PSR. A PSR cell holding 1 (0) implies that the corresponding crosspoint is absent (present), and thus A_{ij} = 0 (1), i.e., the content of the PSR cell in this step is opposite to the entry A_{ij}.

Property 3.

If the j-th row of the matrix AP_F contains all 0's, then the product line P_j is diagnosed as having an s-a-1 fault.

Proof. If the j-th row of the matrix AP_F contains all 0's, two cases can be identified: either all crosspoints of P_j at the AND plane are absent, or P_j has an s-a-1 fault. Since both faults are repaired by spare product lines, in order to simplify the fault location process, we would rather treat the multiple G-faults in P_j as an s-a-1 fault than distinguish these faults. □

Once the *stuck-at* faults have been repaired, the *crosspoint* faults are identified as follows. Let matrix $AP = [a_{ij}]_{p\times r}$ be the personality matrix for the AND array, where a_{ij}'s are defined as the same as A_{ij} in (4.1). The *crosspoint* faults are located by XORing both AP and AP_F.

$$CAP = AP \oplus AP_F = [\, a_{ij} \oplus A_{ij} \,]_{p\times r} \tag{4.2}$$

Any non-zero entry indicates either a G- or S-fault at that position. In fact, both faults can be precisely distinguished by checking the matrix AP, i.e., let CAP(i,j)=1, if a_{ij}=1, the fault is a G-fault; otherwise, it is an S-fault. In order to efficiently utilize the spare lines, both G- and S- faults are not repaired at this time.

4.1.3 Identify and Repair Faults in the OR Plane

The faults in the OR plane include: (1) *stuck-at* faults at output lines; (2) *bridging* faults at adjacent output lines, adjacent product lines, and crossing product and output lines; and (3) A- and D- faults.

By controlling the product lines and observing the output lines, the faults in the OR plane can be detected easily. Figure 4.4 shows that we first apply the test patterns to product lines by assigning (R,W)=(0,1), i.e., the PSR is set to the "write" mode and the ISR is the "read" mode. (But the contents read from the ISR are ignored.) We then assign Vdd1 = "0"; and shift all 0's to the PSR.

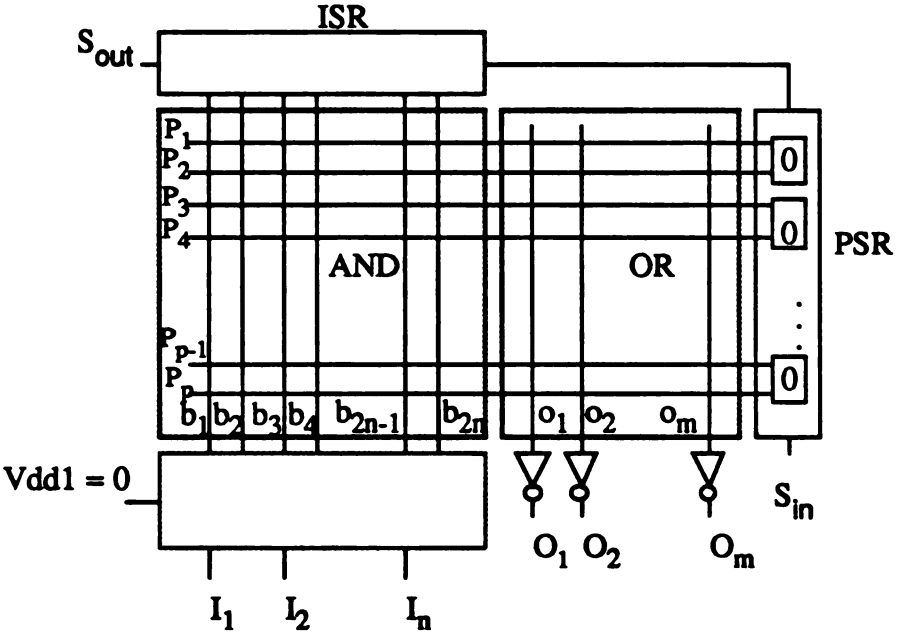


Figure 4.4 Identify s-a-1 Faults at Output Lines.

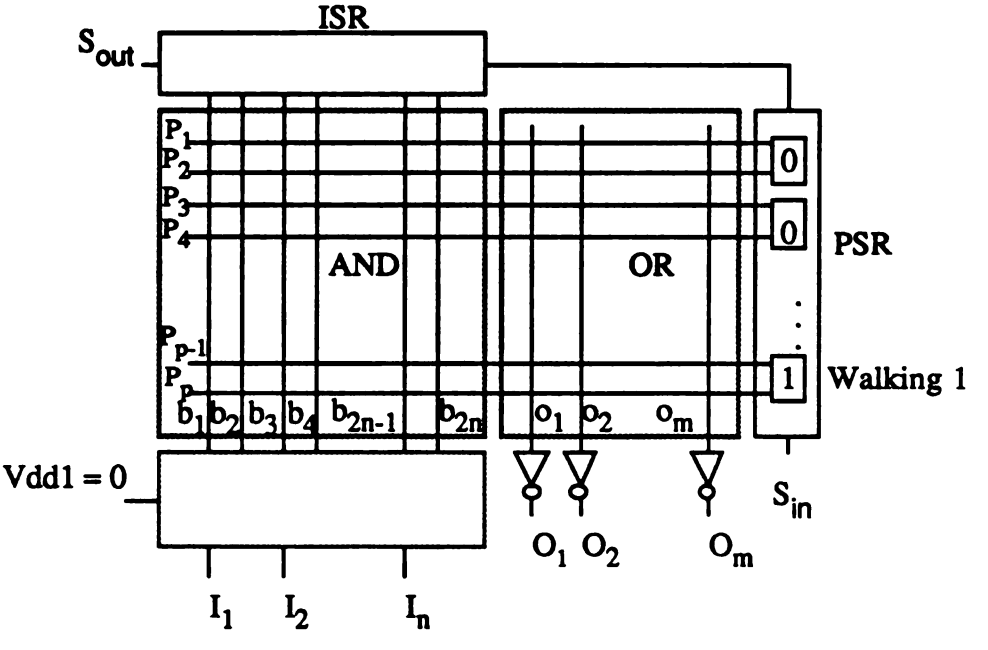


Figure 4.5 Identify Bridging Faults; Outputs s-a-1 Faults; and A- and D- Faults.

The result is that the internal lines of the outputs (without the inverting buffers) are expected to be all 1's, or the output signal lines (with the inverting buffer) are expected to be all 0's. Therefore, an unexpected 1 in an output line indicates that the line has either an s-a-1 fault or a BRPO fault.

Property 4.

Type BRPO faults are equivalent to the bridged output lines with s-a-1 faults and the bridged product lines with s-a-0 faults.

Proof. An s-a-1 faulty output line can be diagnosed by applying all 0's to the product lines. Since the bridged product line has no *stuck-at* fault, the logic 0 in the bridged product line will force the bridged output lines to have s-a-1 faults. On the other hand, since the faulty output lines are repaired by disconnecting them from the SOSC and connecting them to Ground, the bridged product lines are thus forced to be *stuck-at*-0. □

Figure 4.5 illustrates the process of locating the s-a-0 faults at outputs; the *bridging* faults: BRPP, BROO, and BRPO faults; and A- and D-faults. Similar to the process shown in Figure 4.3, a walking "1" pattern is applied to the PSR after the s-a-1 faults have been located and repaired. All the product lines but one are disabled at a time. The status of the crosspoints in each enabled product line are recorded and form the following matrix.

$$OP_F = \begin{bmatrix} B_{11}\, B_{12} \ldots B_{1m} \\ B_{21}\, B_{22} \ldots B_{2m} \\ \ldots\ldots \\ \ldots\ldots \\ B_{p1}\, B_{p2} \ldots B_{pm} \end{bmatrix} \qquad (4.3)$$

where B_{ij}'s are defined as the same as A_{ij} in (4.1).

Property 5.

If the j-th column of the matrix OP_F contains all 0's, then the output line O_j is diagnosed as having an s-a-0 fault.

Proof. The j-th column containing all 0's implies that either O_j has an s-a-0 fault, or all crosspoints at O_j are missed. Both faults are repaired by spare output lines. For simplicity, the multiple D-faults in O_j are diagnosed as O_j with an s-a-0 fault. □

Property 6.

If the j-th row of the matrix OP_F contains all 0's, then the product line P_j is diagnosed as having an s-a-0 fault.

Proof. The j-th row containing all 0's implies that either the product line P_j has an s-a-0 fault, or all crosspoints of P_j in the OR plane are missed. Similarly, for simplicity, the multiple D-faults at P_j are diagnosed as P_j with an s-a-0 fault. □

Property 7.

Type BRPP faults are equivalent to the bridged product lines with s-a-0 faults.

Proof. Recall that a walking "1" pattern was applied after the s-a-1 faults at the product lines had been located and repaired. If we assume that the only "1" is applied to P_j with 0's to all other product lines, the wired-AND *bridging* fault model will force both P_j and P_{j+1} to be bridged to 0. As a result, the product line P_j will not be enabled as it should be. The outputs are then observed as having all 0's, i.e., the j-th row of the matrix OP_F contains all 0's. By Property 6, the product line P_j is diagnosed as having an s-a-0 fault. Similarly, the same argument can be applied to the other bridged product lines, and those lines are diagnosed as having s-a-0 faults. □

The type BROO *bridging* faults are identified by checking the matrix OP_F. When both the j-th and (j+1)-th columns of the matrix OP_F are the same, two cases are identified: either having multiple *crosspoint* faults, or a *bridging* fault at both adjacent output lines, O_j and O_{j+1}. In fact, the BROO *bridging* faults can be distinguished by checking if the bit patterns of the j-th column of the matrix OP_F is the same as the bit patterns obtained from ANDing both the j-th and (j+1)-th columns of the matrix OP.

Once the *stuck-at* faults and *bridging* faults have been repaired, the *crosspoint* faults are identified as follows. Let matrix $OP = [b_{ij}]_{p\times m}$ be the personality matrix for the OR array, where b_{ij} is defined the same as A_{ij} in (4.1). The *crosspoint* faults are located by XORing both OP and OP_F

$$COP = OP \oplus OP_F = [\, b_{ij} \oplus B_{ij} \,]_{p\times m} \tag{4.4}$$

Similarly, any non-zero entry indicates either a D- or A- fault at that position. Both fault types can be precisely identified by checking the matrix OP, i.e., let COP(i,j)=1, if b_{ij}=1, the fault is a D-fault; otherwise, it is an A-fault.

After all the *crosspoint* faults are located, an algorithm for efficiently utilizing spare lines is needed.

4.1.4 Repair Crosspoint Faults

According to the repair rules listed in Table 3.2, both G- and S- faults can be repaired either by spare input bit lines, and/or product lines. Similarly, both D- and A-faults can be repaired either by spare output lines and/or product lines. By concatenating both matrices CAP and COP as the fault map, a spare allocation algorithm developed in [31] can be used to efficiently repair *crosspoint* faults.

4.2 Fault Diagnosis and Repair Algorithm

Based on the fault location and repair process, a fault diagnosis and repair algorithm is summarized in Appendix 2.

Two examples are given in this section to demonstrate the proposed fault diagnosis and repair process. The results will show that the proposed fault diagnosable design achieves a full diagnosability of all single and multiple *stuck-at* faults, *bridging* faults, and *crosspoint* faults.

Consider a fault-tolerant PLA, as shown in Figure 4.6 (a). The PLA consists of 4 input lines, 4 product lines, and 3 output lines, i.e., n=4, m=3, p=4, and r=8. In addition, 2 spare input bit lines, 2 spare product lines, and 2 output lines are also added for the repair of the defective chip, i.e., s_n=2, s_p=2, and s_m=2. The personality of the PLA can be described by the following cubical representation,

```
1-01    010
-101    101
10--    011
00-1    110
```

Both matrices AP and OP are generated according to the personality of the fault-free PLA in Figure 4.6 (a) as follows,

$$AP = \begin{bmatrix} 01001001 \\ 00011001 \\ 01100000 \\ 10100001 \end{bmatrix} \quad \text{and} \quad OP = \begin{bmatrix} 010 \\ 101 \\ 011 \\ 110 \end{bmatrix} \qquad (4.5)$$

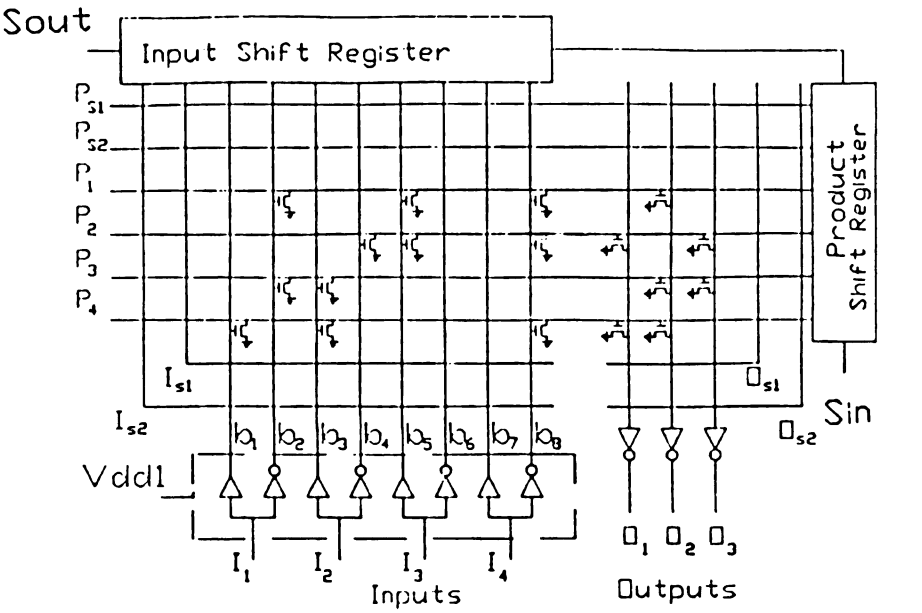


Figure 4.6 Examples in the Fault-Diagnosable PLA Design: (a) Fault-free PLA; and (b) Faulty PLA.

4.2.1 Example 1

Consider a faulty PLA, as shown in Figure 4.6 (b), in which the following faults are applied to the fault-free PLA,

b_2 : s-a-0, P_1: s-a-1, O_2: s-a-1,

AP(3,3):G-fault, OP(3,1): A-fault, I_{s1}: s-a-1.

When the scan patterns are applied, the observed scan-out patterns match the applied scan-in patterns. Thus, no fault is detected in the added circuits.

For Step B, after the input patterns (I_1,I_2,I_3,I_4)=(0,0,0,0) and (1,1,1,1) are applied, we obtain

EP0=(0,0,0,1,0,1,0,1),

and

EP1=(1,0,1,0,1,0,1,0).

This results in b_2 having an s-a-0 fault, and the faulty bit line being repaired by the spare input bit line I_{s1}. The number of spare bit lines becomes s_n=1.

In order to diagnose the possible faults in the spare input lines, Step B is proceeded again, and both EP0 and EP1 are re-generated as follows,

EP0=(0,1,0,1,0,1,0,1),

and

EP1=(1,1,1,0,1,0,1,0).

This causes the bit position of b_2 (now is I_{s1}) to have an s-a-1 fault, and the line is then repaired by the spare input line I_{s2}. The number of spare input bit lines becomes s_n=0, i.e. no more spare bit lines are available. After the faulty line has been repaired, Step B is again repeated and both EP0 and EP1 are finally obtained as

EP0=(0,1,0,1,0,1,0,1),

and

EP1=(1,0,1,0,1,0,1,0).

This shows that no *stuck-at* fault is identified in the bit lines.

For Step C, when all 0's are applied to bit lines b_i's, we get the contents of the product lines as $(P_1,P_2,P_3,P_4)=(1,1,1,1)$, which shows no s-a-0 fault in P_i. Therefore, the matrix AP_F is generated as follows,

$$AP_F = \begin{bmatrix} 00000000 \\ 00011001 \\ 01000000 \\ 10100001 \end{bmatrix}$$

Since the first row contains all 0's, an s-a-1 fault at P_1 is identified. Therefore, P_1 is repaired by the spare product line P_{s1}, and the number of spare product lines becomes $s_p=1$. Once faults are repaired, the fault location process is then repeated to start from Step B, and the matrix AP_F is re-generated as

$$AP_F = \begin{bmatrix} 01001001 \\ 00011001 \\ 01000000 \\ 10100001 \end{bmatrix}$$

The matrix AP_F shows that no s-a-1 fault at the product lines is identified. Finally, we get the matrix CAP as follows,

$$CAP = \begin{bmatrix} 00000000 \\ 00000000 \\ 00100000 \\ 00000000 \end{bmatrix}$$

The matrix indicates that the bit position CAP(3,3) may have either a G- or S-fault. In fact, the bit position has a G-fault because AP(3,3)=1 in (4.5). This G-fault will be repaired later.

For Step D, 0's are applied to all product lines, so we get the output lines as $(O_1,O_2,O_3)=(0,1,0)$. The output line O_2 is diagnosed as having an s-a-1 fault, and repaired by a spare output line O_{s1}. The number of spare output lines becomes $s_m=1$.

Once the faults are repaired, the entire fault location proceeds to start from Step B. We then obtain $(O_1,O_2,O_3)=(0,0,0)$ as expected. This is followed by the generation of the matrix OP_F,

$$OP_F = \begin{bmatrix} 0\,1\,0 \\ 1\,0\,1 \\ 1\,1\,1 \\ 1\,1\,0 \end{bmatrix}$$

The matrix shows that neither an s-a-0 fault at the output lines and product lines, nor the *bridging* faults (BROO), is identified. Therefore, we generate the matrix COP as

$$COP = \begin{bmatrix} 0\,0\,0 \\ 0\,0\,0 \\ 1\,0\,0 \\ 0\,0\,0 \end{bmatrix}$$

However, this matrix shows that the bit position COP(3,1) may have either a D-fault or an A-fault. In fact, the bit position has an A-fault due to OP(3,1)=0 in Equation (4.5).

For Step E, if we merge both matrices CAP and COP, we get

$$\begin{bmatrix} \text{CAP} & \text{COP} \\ 00000000 & 000 \\ 00000000 & 000 \\ 00100000 & 100 \\ 00000000 & 000 \end{bmatrix}$$

Taking this fault map, the spare allocation algorithm finds a solution such that a spare product line is used to repair both *crosspoint* faults. After the faults are repaired, the entire location process is then proceeded again. The result shows that no further fault is identified, and the process is done.

4.2.2 Example 2

Consider the application of the following *bridging* faults to the PLA of Figure 4.6 (a),

(1) at the bit lines b_2 and b_3;

(2) at the product line P_2 and the output line O_2; and

(3) at the product lines P_3 and P_4, where P_4 has also an s-a-0 fault.

The fault diagnosis process proceeds as follows. First, when the scan pattern is applied, the observed scan-out patterns match the applied scan-in pattern. Thus, no fault is detected in the added circuits.

For Step B, after the input patterns (I_1,I_2,I_3,I_4)=(0,0,0,0) and (1,1,1,1) are applied, we obtain

EP0=(0,0,0,1,0,1,0,1),

and

EP1=(1,0,0,0,1,0,1,0).

This results in both bit lines b_2 and b_3 having s-a-0 faults. The faults are repaired by the spare input bit lines I_{s1} and I_{s2}, respectively. The number of spare bit lines becomes to s_n=0.

In order to diagnose the faults in the spare lines, Step B runs again, both EP0 and EP1 are re-generated as follows,

EP0=(0,1,0,1,0,1,0,1),

and

EP1=(1,0,1,0,1,0,1,0).

This shows that no *stuck-at* fault at the bit lines is identified.

For Step C, when 0's are applied to all bit lines b_i's, we obtain the contents of the product lines as $(P_1,P_2,P_3,P_4)=(1,1,0,0)$, which shows that both P_3 and P_4 are diagnosed as having s-a-0 faults. Therefore, both lines are repaired by the spare product lines P_{s1} and P_{s2}, and the number of spare product lines becomes $s_p=0$. The location process then proceeds again, and $(P_1,P_2,P_3,P_4)=(1,1,1,1)$ as expected. Then, we generate the matrix AP_F as follows,

$$AP_F = \begin{bmatrix} 01001001 \\ 00011001 \\ 01100000 \\ 10100001 \end{bmatrix}$$

The matrix shows that no s-a-1 fault at the product lines is identified. Since both matrices AP and AP_F are the same, i.e., the matrix CAP is a zero-matrix, neither a G-fault nor an S-fault at the AND plane is identified.

For Step D, when 0's are applied to all product lines, we get the output lines as $(O_1,O_2,O_3)=(0,1,0)$, i.e. the output line O_2 is diagnosed as having an s-a-1 fault, and is repaired by a spare output line O_{s1}. The number of spare output lines becomes $s_m=1$.

Once the faults are repaired, the entire fault location proceeds again to start from Step B, and $(O_1,O_2,O_3)=(0,0,0)$ as expected. Then, followed by the generation of the matrix OP_F,

$$OP_F = \begin{bmatrix} 010 \\ 000 \\ 011 \\ 110 \end{bmatrix}$$

The matrix shows that no s-a-0 fault at the output lines is identified. But, an s-a-0 fault is identified at P_2 because of all zero's in the second row of the matrix OP_F. Therefore, it should be repaired by a spare product line. Unfortunately, since s_p=0 implies that no more spare lines are available for repair, the defective chip is claimed to be unrepairable. The fault diagnosis process is then terminated.

4.2.3 Discussion

Example 1 shows that a defective chip can be repaired by the assigned spares. However, the defective chip in Example 2 is unrepairable.

Example 2 also shows that the *bridging* fault at the adjacent bit lines (or type BRII) are diagnosed as both bit lines having s-a-0 faults (Property 2). The *bridging* fault at the output line and product line (or type BRPO) is diagnosed as the output line with an s-a-1 fault and the product line with an s-a-0 fault (Property 4). Finally, the *bridging* fault at the adjacent product line, where one bridged product line has an s-a-0 fault, is diagnosed as both product lines with s-a-0 faults (Property 1).

4.3 Test Chip in Field Use

Due to lack of controllability and observability, PLA testing, particularly for large chips, has been recognized as a very difficult problem. To alleviate such a problem, the design of easily testable PLAs have been popularly implemented [9,25,59]. The key to the easily testable PLA design is the use of additional hardware to enable only one product line at a time to increase the testability [54].

In Section 3.4, an easily testable PLA design has been presented and shown in Figure 3.21. The design modifies the fault diagnosable PLA design, but requires no additional hardware.

That PLA is tested as follows: Basically, a walking one pattern is applied to the PSR to enable only one product line. For each enabled product line, the use of the main test pattern and auxiliary patterns proposed in [6] can then detect single and multiple faults in either the PSR, or the PLA itself.

4.4 Summary

A fault diagnosis and repair algorithm has been presented. The algorithm shows that the proposed fault diagnosable PLA design achieves a full diagnosability of single and multiple *stuck-at*, *bridging*, and *crosspoint* faults. A test procedure is also presented for a packaged PLA chip. It should be noted that the fault-tolerant PLA is repaired by the use of laser programming techniques. Our diagnosis and repair process basically implements the scheme that locates *stuck-at* and *bridging* faults and then repairs them.

CHAPTER 5

Yield Analysis

This chapter analyzes the effects of adding redundancy to the design of fault-tolerant PLAs. In theory, a higher probability of repair can be achieved if a larger number of spares is added. However, since the added redundancy and the associated circuitry are also susceptible to defects, too much redundancy may have a "diminishing" effect on the chip. Therefore, it is not always guaranteed that the additional redundancy improves the overall probe yield.

In this chapter a yield model for the design of fault-tolerant PLAs is presented and simulated in Section 5.1 and 5.2. Based on the yield model, the optimal redundancy that provides the maximal chip yield is discussed in Section 5.3.

5.1 Yield Model

Productivity of chips is reduced both by gross imperfections and from faults caused by random defects in the photolithography and materials [46]. The productivity of the redundant chip is also strongly dependent upon the percentage of uncorrectable defect area of the chip. In the design of fault-tolerant PLAs, the defects in the AND and OR planes can be repaired with the spare lines. Other faults, such as the defects in decoders, power lines, clock lines, and control circuits are considered as fatal errors and are not correctable by redundancy. As a result, the net yield with

redundancy, Y_{NET}, is the product of the imperfection yield Y_{GI}, correctable yield Y_{CRD}, and uncorrectable yield Y_{URD}, i.e.,

$$Y_{NET} = Y_{GI} \times Y_{CRD} \times Y_{URD} \tag{5.1}$$

Since the yield Y_{GI} depends on a fabrication process which is not available at this time, for brevity, we consider the relative net yield (Y_{NET}/Y_{GI}), or effective yield Y_{eff}, i.e.,

$$Y_{eff} = Y_{URD} \times Y_{CRD} \tag{5.2}$$

For the statistics of the fabrication defects we can adopt one of the models suggested in the literature, such as Poisson, general negative binomial, or binomial statistics. In this work, the Polya-Eggenberger distribution, mixed Poisson statistics using gamma distribution as a mix function [12,28,29,55,62,64], were employed to define the random defects for the fabrication of chips. The probability of having x faults on a chip for this distribution is

$$P(X=x) = \frac{\Gamma(x+\alpha) \quad (\lambda/\alpha)^x}{x!\,\Gamma(\alpha)\,(1+\lambda/\alpha)^{\alpha+x}} \tag{5.3}$$

where α is a clustering parameter that depends on the defect density variation [55] and λ is the average number of faults per chip. In this work, it is assumed that the correctable defect area of FTPLA includes only the AND/OR arrays and the spare lines; the remaining chip area is uncorrectable.

5.1.1 Correctable Random Effect Yield, Y_{CRD}

The correctable yield, Y_{CRD}, is affected by the array yield, Y_{array}, and the area penalty for redundancy, where the area penalty is the ratio of the chip area without redundancy (A_{NR}) to that with redundancy (A_R). The yield Y_{CRD} is expressed as

$$Y_{CRD} = Y_{array} \times (A_{NR} / A_R) \tag{5.4}$$

In the previous work [62,64], the array yield was defined as the probability that the number of faults will be less than, or equal to, the total number of spare lines s, i.e.,

$$Y_{array} = P(x \le s) = \sum_{x=0}^{s} \frac{\Gamma(x+\alpha) \quad (\lambda/\alpha)^x}{x!\,\Gamma(\alpha)\,(1+\lambda/\alpha)^{\alpha+x}} \tag{5.5}$$

However, the above model ignores the fact that the faults in the AND array can be repaired only by either spare input or product lines, but not by the spare output line. Also, the faults in the OR array cannot be repaired by the spare input lines. Considering the above fact, a more accurate array yield has been proposed recently [12]. The yield is expressed as

$$\begin{aligned} Y_{array} &= P(X_1 \le s_{AND})\, P(X_2 \le s_{OR}) \\ &= \sum_{\substack{x_1=0 \\ x_1+x_2 \le s}}^{s_{AND}} \sum_{x_2=0}^{s_{OR}} \left[\frac{\Gamma(x_1+\alpha)\,(A_1 d/\alpha)^{x_1}}{x_1!\,\Gamma(\alpha)\,(1+A_1 d/\alpha)^{\alpha+x_1}}\right]\left[\frac{\Gamma(x_2+\alpha)\,(A_2 d/\alpha)^{x_2}}{x_2!\,\Gamma(\alpha)\,(1+A_2 d/\alpha)^{\alpha+x_2}}\right] \end{aligned} \tag{5.6}$$

where $s_{AND} = s_n+s_p$, $s_{OR} = s_p+s_m$, $s = s_n+s_p+s_m$ (s_n, s_p, and s_m are the number of spare input, product, and output lines, respectively). A_1 and A_2 are the chip areas of AND and OR arrays, respectively. We assume that the defect density d and the cluster parameter α in both arrays are identical.

Recall that, as the fault diagnosis process discussed in the previous chapter, both *stuck-at* faults and *bridging* faults are repaired by the corresponding spares. For instance, stuck-at faulty product lines must be repaired by spare product lines. On the other hand, crosspoint faults are optimally repaired by the available spare lines that are determined by the spare allocation algorithm. In other words, all the faults are repaired by either spare input, product, or output lines. Therefore, following the proposed repair rules and fault diagnosis algorithm, a new yield model for the array yield is presented.

A fault-tolerant PLA is said to be *repairable* if the spare lines can cover all faults. Otherwise, it is *unrepairable*. In other words, the PLA is repairable if various types of spares are sufficient to repair the corresponding types of faulty lines. Let $P_w(s_w)$ be the probability that the number of w-type faulty lines is less than or equal to s_w, where w = n for bit line, w = m for output line, and w = p for product line. The array yield is then expressed by the product of these probabilities, i.e.,

$$Y_{array} = P_n(s_n) \times P_p(s_p) \times P_m(s_m) \tag{5.7}$$

In general, the probability that has exactly i faulty product lines out of the total of $p+s_p$ product lines can be described by the Binomial Distribution function, i.e.,

$$P_p(X = i) = C_i^{p+s_p} \; q_p^{\,i} (1 - q_p)^{p+s_p-i} \tag{5.8}$$

where q_p is the failure rate of the product lines. Thus, the probability $P_p(s_p)$ is expressed as

$$P_p(s_p) = P(X \le s_p) = \sum_{i=0}^{s_p} C_i^{p+s_p} \; q_p^{\,i} (1 - q_p)^{p+s_p-i} \tag{5.9}$$

Similarly, the probabilities $P_m(s_m)$ and $P_n(s_n)$ are

$$P_m(s_m) = P(X \le s_m) = \sum_{i=0}^{s_m} C_i^{m+s_m} \; q_m^{\,i} (1 - q_m)^{m+s_m-i} \tag{5.10}$$

$$P_n(s_n) = P(X \le s_n) = \sum_{i=0}^{s_n} C_i^{2n+s_n} q_n^{\,i} (1 - q_n)^{2n+s_n-i} \tag{5.11}$$

Therefore, substituting the array yield in Equation (5.7) to Equation (5.4), the correctable random effect yield Y_{CRD}, is

$$Y_{CRD} = P_n(s_n) \times P_p(s_p) \times P_m(s_m) \times A_{NR}/A_R \tag{5.12}$$

5.1.2 Uncorrectable Random Effect Yield, Y_{URD}

The percentage of uncorrectable defect area is one of the key factors in determining the effectiveness of redundancy. The uncorrectable yield is defined as [12]

$$Y_{URD} = (1 + \lambda \times (A_{UNC} / A_{SUS}) / \alpha)^{-\alpha} \qquad (5.13)$$

where A_{UNC} and A_{SUS} are the uncorrectable defect-susceptible area and the total defect-susceptible area, respectively. In general, the random defect yield is very sensitive to the percentage of the uncorrectable defect area. A low percentage of uncorrectable defect area allows one to go higher levels of integration before the yield term falls off significantly.

5.2 Yield Simulation

Although the proposed fault-tolerant PLA has not yet been fabricated and the experimental processing data are not available to precisely determine the failure rates, we may employ the experimental data studied in [48]. This experiment predicts all faults that are likely to occur in a MOS integrated circuit or subcircuit.

The study shows that, of the original 4800 defects, only 476 actually produce significant faulty behaviors at the circuit level. They can classified as follows: 72 crosspoint faults, 388 stuck-at or bridging faults, and 16 power line faults. Since the power line faults do not affect the calculation of array yield, they are excluded. According to our repair rules, most crosspoint faults are efficiently repaired by spare product lines. Thus, it is reasonable to assume that the number of crosspoint faults repaired by spare product lines is as many as twice that repaired by spare input lines and by spare output lines. Also, we assume that the stuck-at and bridging faults are uniformly distributed to each type of line. Based on these assumptions, of the 460 faults, 147 are contributed to bit lines, 166 to product lines, and another 147 to output

lines. In other words, the study shows that both bit lines and output lines have the same failure rate because their structures are virtually the same, i.e., $q_n = q_m$, but the failure rate q_p is nearly 12% higher than q_n, i.e., $q_p = 1.12\, q_n$.

Let $q_n = q_m = q$, and thus $q_p = 1.12\, q$. The failure rate q is generally obtained from the statistics in the fabrication process and manufacturing process. In this study, however, the failure rate is roughly estimated from the following calculation. The basic concept is that a non-redundant PLA design is conceptually identical to the redundant PLA design with the spares $(s_n, s_p, s_m) = (0,0,0)$. Therefore, the probabilities of having any failures in both designs should be the same. In practice, the former probability can be obtained from Equation (5.3) with x = 0, i.e.,

$$P_{NR} = P(x=0) = (1 + \lambda / \alpha)^{-\alpha} \tag{5.14}$$

and the latter probability is the product of Y_{URD}, in Equation (5.13), and Y_{CRD} with $(s_n, s_p, s_m) = (0,0,0)$, in Equation (5.12), i.e.,

$$\begin{aligned} P_R &= Y_{CRD}|_{(s_n, s_p, s_m) = (0,0,0)} \times Y_{URD} \\ &= [P_n(0)P_p(0)P_m(0) \times A_{NR}/A_R] \times [1+(\lambda/\alpha)(A_{UNC}/A_{SUS})]^{-\alpha} \end{aligned} \tag{5.15}$$

For a redundant design with no redundancy, $A_{NR} = A_R$, and, by Equations (5.9)-(5.11), $P_n(0) = (1-q)^{2n}$, $P_m(0) = (1-q)^m$, and $P_p(0) = (1-1.12q)^p$, Equation (5.15) can be written as

$$P_R = [(1-q)^{2n+m}(1-1.12q)^p] \times [1+(\lambda/\alpha)(A_{UNC}/A_{SUS})]^{-\alpha} \tag{5.16}$$

By equating both P_{NR} in Equation (5.14) and P_R in Equation (5.16), we obtain

$$(1+\lambda/\alpha)^{-\alpha} = [1+(\lambda/\alpha)(A_{UNC}/A_{SUS})]^{-\alpha}\, [(1-q)^{2n+m}(1-1.12q)^p] \tag{5.17}$$

If the parameters α and λ, and the area ratio A_{UNC}/A_{SUS} are given, then, we should be able to solve Equation (5.17) for q.

For example, consider a (50,190,67)-PLA; according to the floor plane shown in Figure 3.13, the area ratio is calculated as $(A_{UNC}/A_{SUS}) = 0.2038$. Let $\alpha = 2$ [62] and consider the case of $\lambda = 4$, Equation (5.17) results in $q = 0.00398$, i.e., the failure rates $q_n = q_m = 0.00398$ and $q_p = 0.00446$.

The failure rates are subject to the number of average faults. For various average numbers of faults, Figure 5.1 illustrates the correctable random effect yield Y_{CRD}, uncorrectable random effect yield Y_{URD}, and the effective yield Y_{eff} for the (50,190,67)-PLA with $(s_n,s_p,s_m) = (3,4,2)$. For the case of $\lambda = 4$, $Y_{CRD} = 88.79\%$, $Y_{URD} = 54.13\%$, and $Y_{eff} = 48.06\%$. This shows that the chip yield for the redundant design is much higher than the 11.1% yield for the nonredundant design.

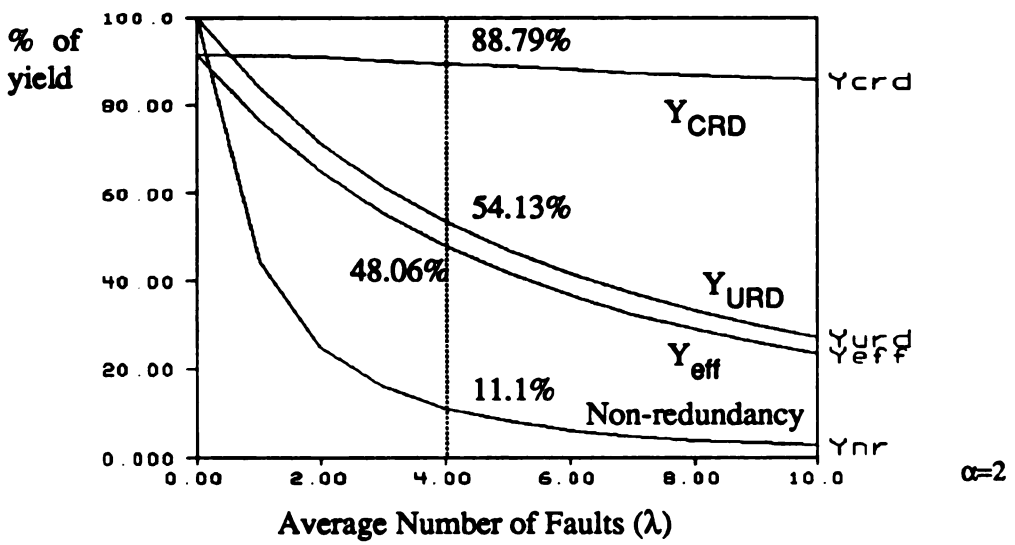


Figure 5.1 Yields for (50,190,67)-PLA with Redundancy $(s_n,s_p,s_m)=(3,4,2)$.

Figure 5.2 illustrates the effects of adding redundancy. Figure 5.2 (a) plots the correctable yield Y_{CRD} versus the number of spare product lines, where s_n=3, s_m = 2, and s_p is varied from 0 to 10. The plot shows that, as the number of spare product lines increases, the array yield Y_{array} increases, but the area ratio A_{NR}/A_R decreases. As a result, the overall Y_{CRD} is increased initially, but decreased as the number of spare product lines increases. For example, Y_{CRD} = 86.10% for s_p = 2, Y_{CRD} = 89.00% for s_p = 3, but Y_{CRD} = 88.79% for s_p = 4.

Figure 5.2 (b) plots various yield simulations versus the number of spare product lines. The plots also show that the yield Y_{eff} = 47.93% for (s_n,s_p,s_m) = (3,3,2), and increases to 48.06% for (s_n,s_p,s_m) = (3,4,2), but drops to 47.82% for (s_n,s_p,s_m) = (3,5,3). The results show that the additional redundancy may not improve the overall yield.

Figure 5.2 (c) plots the effective yields for the spare assignments (4,6,4), (3,4,2), and (2,2,1) versus the average number of faults. For λ = 0, the effective yields for (4,6,4), (3,4,2), and (2,2,1) are 83.74%, 89.23%, and 93.81%, respectively. (It should be noted that the yield does not reach 100% due to the area penalty). For λ = 1, the yields for the spare assignments are respectively 71.08%, 75.11%, and 77.57%, but for λ = 10, the yields are respectively 24.36%, 24.33%, and 20.29%. This implies that less redundancy is better if the average number of faults is smaller, but, for larger numbers of faults, more redundancy may produce a higher chip yield.

Figure 5.2 has provided significant evidence that the additional redundancy may not improve the yield. This motivates the study of finding optimal redundancy in the proposed fault-tolerant PLA design.

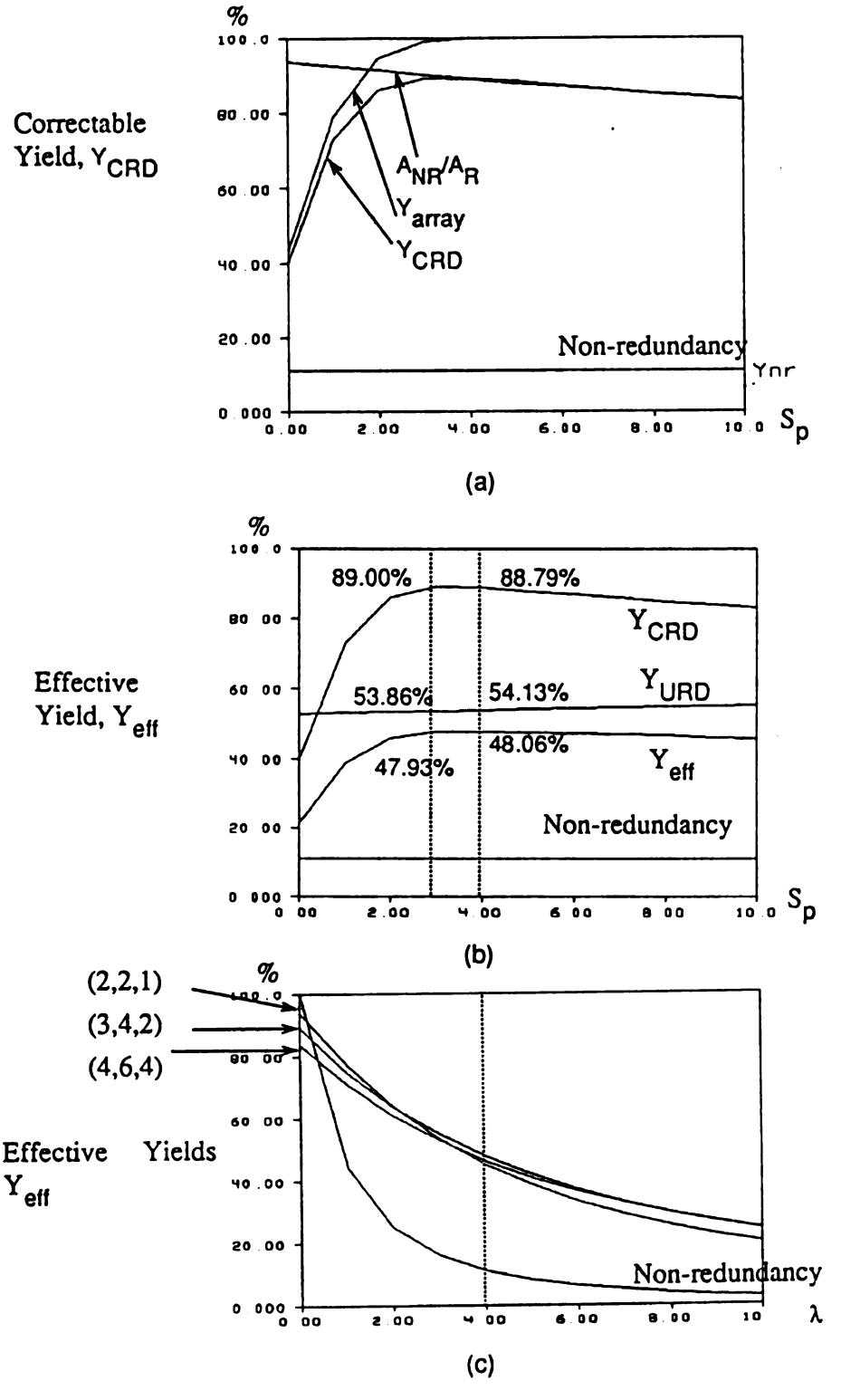


Figure 5.2 Yield Analysis for (50,190,67)-PLA with α=2:
(a) Correctable Yield, (λ = 4); (b) Effective Yield, (λ = 4); and
(c) Yields for Various Spare Line Assignments.

5.3 Optimal Redundancy

Integrated circuit manufacturers find it highly desirable to be able to predict the yield loss before a chip is fabricated, and to expect to maximize the probe yield, and thus maximize profits. In this section, an efficient way to determine the optimal redundancy in the proposed fault-tolerant PLA design is presented.

As more redundancy is added to redundant PLAs, both yield and productivity increase; however, the redundant spare lines inflate die size and reduce the number of chips per wafer. Figure 5.2 has evidently shown that, as the number of spare lines increases, the array yield increases, but the area ratio A_{NR}/A_R drops. As the redundancy increases, a point will eventually be reached where optimum yield is obtained.

The optimal redundancy problem can be expressed by the following nonlinear integer optimization problem:

$$\text{Maximize} \quad Y_{eff} = Y_{URD} \times [\, P_n(s_n)P_p(s_p)P_m(s_m) \times A_{NR}/A_R \,]$$

$$\text{Subject to} \quad 0 < s_n \le 2n;\ 0 < s_p \le p;\ 0 < s_m \le m.$$

Finding the parameters s_n, s_p, and s_m is obviously not an easy task, even though the parameters are integers. However, since the yield calculation is relatively simple, the optimal solution can be easily formed if the bounds of parameters (s_n, s_p, s_m) are narrowed.

Consider the Binomial distribution

$$P(X = i) = C_i^N q^i (1 - q)^{N-i} \tag{5.18}$$

where P(i) is the probability which has exactly i faulty lines from the total N lines, and q is the failure rate. The expected value μ and standard deviation σ of this distribution are $\mu = Nq$, and $\sigma = \sqrt{Nq(1-q)}$, respectively.

An empirical rule is applied to define the upper and lower bounds of the parameters (s_n,s_p,s_m). Let $x = \mu+3\sigma$; the ceiling function $\lceil x \rceil$ is defined as the upper bound and the floor function $\lfloor x \rfloor$ as the lower bound. Therefore,

$$\lfloor k_n \rfloor \le s_n \le \lceil k_n \rceil, \text{ where } k_n = 2nq+3\sqrt{2nq(1-q)}\ , \tag{5.19}$$

$$\lfloor k_p \rfloor \le s_p \le \lceil k_p \rceil, \text{ where } k_p = p(1.12q)+3\sqrt{p(1.12q)(1-1.12q)}\ , \tag{5.20}$$

$$\lfloor k_m \rfloor \le s_m \le \lceil k_m \rceil, \text{ where } k_m = mq+3\sqrt{mq(1-q)}\ . \tag{5.21}$$

Note that the term N in both μ and σ for the product lines includes the number of the original and spare product lines, i.e., $N = P+s_p$. However, for simplicity, we will only consider $N = p$ because the number of spare line is far smaller than p, i.e., $s_p << p$. Similarly, both s_n and s_m are omitted in Equations (5.19) and (5.21).

Consider the (50,190,67)-PLA with λ=4. In the previous discussion, we have q=0.00398. Equations (5.19)-(5.21) result in k_n=2.29, k_p=3.6, and k_m=1.81, i.e., $2 \le s_n \le 3$, $3 \le s_p \le 4$, $1 \le s_m \le 2$. Therefore, we may exhaustively calculate the yields for all 8 possible combinations of (s_n,s_p,s_m). Table 5.1 lists the yield

Table 5.1 Yield Simulation for (50,190,67)-PLA.

Redundancy s_n	s_p	s_m	Y_{CRD}	Y_{URD}	Effective Chip Yield
2	3	1	88.34%	53.24%	47.03%
2	3	2	89.45%	53.58%	47.93%
2	4	1	88.13%	53.51%	47.16%
2	4	2	89.24%	53.84%	48.05%
3	3	1	87.88%	53.52%	47.04%
3	3	2	89.00%	53.86%	47.93%
3	4	1	87.67%	53.79%	47.16%
3	4	2	88.79%	54.13%	48.06%

calculations. The results show that the chip yield of 48.06% for (3,4,2) is the maximum. In other words, the assignment (3,4,2) is the optimal redundancy. In order to verify that the assignment (3,4,2) is indeed optimal, it has been calculated the yields for all combinations of (s_n,s_p,s_m), where each parameter is varied from 1 to 5. The results, as listed in Table 5.2, show that the yield of (3,4,2) is, in fact, the highest.

Table 5.2 The Effective Yields for (50,190,67)-PLA

($1 \le s_n, s_m, s_p \le 5$)

s_p = 1

s_n \ s_m	1	2	3	4	5
1	36.36%	37.05%	36.82%	36.51%	36.20%
2	38.15%	38.88%	38.64%	38.32%	37.99%
3	38.15%	38.88%	38.65%	38.33%	38.00%
4	37.92%	38.65%	38.41%	38.09%	37.77%
5	37.66%	38.38%	38.15%	37.84%	37.52%

s_p = 2

s_n \ s_m	1	2	3	4	5
1	43.15%	43.97%	43.69%	43.33%	42.96%
2	45.27%	46.14%	45.85%	45.47%	45.08%
3	45.27%	46.14%	45.86%	45.47%	45.09%
4	44.99%	45.86%	45.58%	45.20%	44.82%
5	44.69%	45.55%	45.27%	44.90%	44.52%

s_p = 3

s_n \ s_m	1	2	3	4	5
1	44.83%	45.68%	45.39%	45.01%	44.62%
2	47.03%	47.93%	47.63%	47.23%	46.82%
3	47.04%	47.93%	47.64%	47.24%	46.83%
4	46.74%	47.64%	47.34%	46.95%	46.55%
5	46.42%	47.31%	47.02%	46.63%	46.24%

s_p = 4

s_n \ s_m	1	2	3	4	5
1	44.95%	45.80%	45.51%	45.12%	44.73%
2	47.16%	48.05%	47.75%	47.35%	46.94%
3	47.16%	48.06%	47.76%	47.35%	46.95%
4	46.86%	47.76%	47.46%	47.06%	46.66%
5	46.54%	47.43%	47.14%	46.75%	46.35%

s_p = 5

s_n \ s_m	1	2	3	4	5
1	44.73%	45.57%	45.28%	44.90%	44.51%
2	46.92%	47.81%	47.51%	47.11%	46.70%
3	46.92%	47.82%	47.52%	47.11%	46.71%
4	46.63%	47.52%	47.22%	46.83%	46.43%
5	46.31%	47.19%	46.90%	46.51%	46.11%

Similarly, for the (100,400,100)-PLA [18,62,64] with λ=4, we obtain q=0.00237, k_n=2.54, k_p=4.15, and k_m=1.70. This results in s_n=2 or 3, s_p=4 or 5, and s_m=1 or 2. The yields are calculated and listed in Table 5.3. The results show that (3,4,2) is the optimal redundancy.

Table 5.3 Yield Simulation for (100,400,100)-PLA.

Redundancy					Effective
s_n	s_p	s_m	Y_{CRD}	Y_{URD}	Chip Yield
2	4	1	91.45%	67.12%	61.38%
2	4	2	92.69%	67.30%	62.38%
2	5	1	91.25%	67.24%	61.35%
2	5	2	92.49%	67.41%	62.35%
3	4	1	91.78%	67.27%	61.74%
3	4	2	93.03%	67.44%	62.74%
3	5	1	91.58%	67.39%	61.72%
3	5	2	92.83%	67.56%	62.71%

5.4 Summary

In this chapter, the yield analysis of fault-tolerant PLA and the optimal spare lines assignment have been presented. Although the total augmented area overhead is nearly 10% to 25% over the original PLA, the results of this study show that the redundant design can enhance the chip yield significantly.

An empirical rule has been applied to simplify the optimization problem for finding the optimal redundancy assignment. The proposed approach of finding optimal redundancy is simple and effective.

Chapter 6

Fault-Tolerant RAM-Based PLAs

The availability of programmable logic devices based on memory cells now allows implementation of "soft" hardware [16], i.e., hardware whose functions can be changed while it resides in the system. With the most current IC component technologies, once a given logic function is implemented in hardware, changing that logic is difficult, requiring modifications to printed-circuit board traces, the addition or replacement of components, and other costly procedures. However, with RAM-based programmable logic, changes can be made to a system's logic functions simply by re-programming the devices.

Recently, the device re-programmability have been exploited in the following applications. Buffalo Products [16] employs a RAM-based programmable logic device for the bus and memory interface and control logic in their More Memory expansion card for PC XT- or AT-compatible systems. In that system, an installation program analyzes system parameters such as bus width, type of card slot, available address spaces, etc., and then loads the appropriate configuration program to match system requirements.

A RAM-based programmable logic device has also been applied to the design of a frame-grabber board of digital imaging system [16]. Basically, the device provides the graphics control, and it interfaces a PC-compatible computer with the video output of such medical equipments as ultrasonic scanners and magnetic resonance imaging systems. To support different video formats from the varying

types of medical instruments, several different configuration programs are available for the device. In this application, each video format is sequentially used for a specific medical instrument, i.e., only one format is used at a time.

During the past decade, several re-programmable logic arrays have been proposed: an electrically programmable and UV erasable implementation of PLA [15], an electrically programmable and erasable PLA [19], and an alterable PLA [35]. These designs allow PLAs to be reprogrammed repeatedly in the same circuit during system prototyping and these PLAs can be re-programmed in different circuits. In this chapter, we focus on the design of RAM-based programmable logic arrays (RBPLAs). An RBPLA is a PLA that takes RAM cells as its crosspoint contacts. The RBPLA is typically used as a programmable device controller [35].

Fault-tolerant PLA design implementing laser programming technique has been presented in Chapter 3 to enhance probe yield in the manufacturing process. However, after the chip is packaged, faults cannot be repaired. In order to efficiently utilize the spare elements resided in PLA chips and to repair faults which may occur either in manufacturing process or in field, the fault-tolerant PLA design implementing the electrically programming technique is motivated.

In the next section, an RBPLA structure is presented. Followed by a fault-tolerant RBPLA design in Section 6.2, and a fault-diagnosis and repair process in Section 6.3. Finally, a chapter summary is given in Section 6.4.

6.1 Basic Structure of an RBPLA

Two types of RAM cells may be implemented: static cell and dynamic cell. Although the array size of the dynamic cell approach is much smaller than that of the static one for the same device density, the need of the control and refresh circuits in the dynamic approach results in a slower speed than the static one in their performance.

In this section, two RBPLA structures with dynamic and static cell memories are discussed. The PLA with dynamic cell memory is referred to as DRBPLA for short, while the one with static cell memory is referred to as SRBPLA.

6.1.1 A DRBPLA Structure

A PLA based on the dynamic cell memory has been proposed in [35]. The DRBPLA, as shown in Figure 6.1, allows users to reprogram the PLA as many times as needed. The PLA consists of two major parts: (1) the basic PLA functions, with the AND and OR arrays, i.e., the A-cells (corresponding to crosspoints of a conventional PLA) and input/output lines; and (2) the control logic needed to program the logic of the PLA and to maintain (refresh) the state of the dynamic storage cells, such as shift registers, R/W control circuits, and the refresh-cells (R-cells).

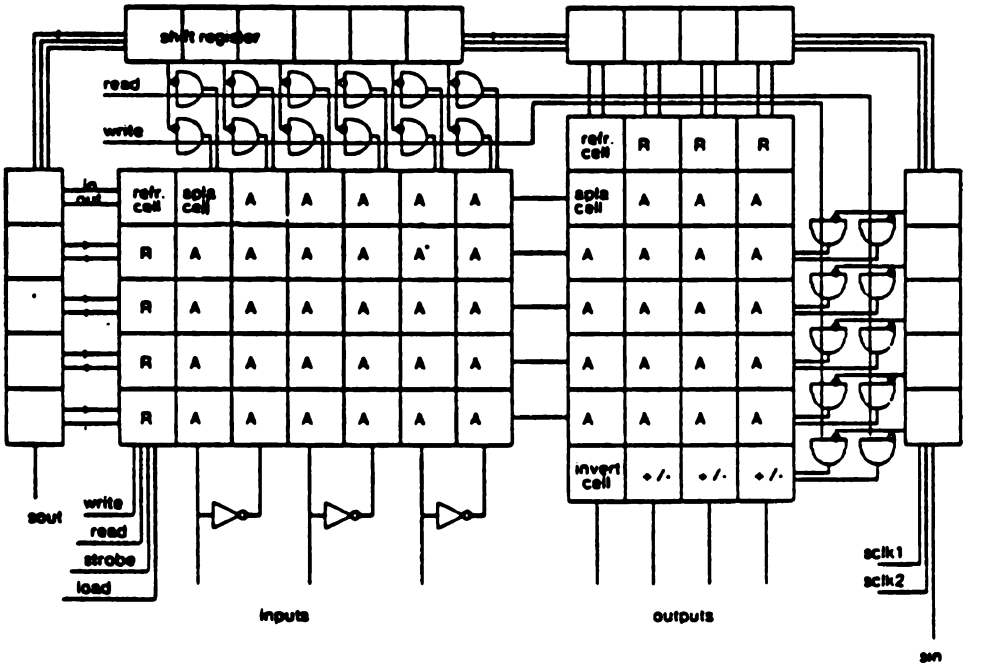


Figure 6.1 The Structure of a DRBPLA [35].

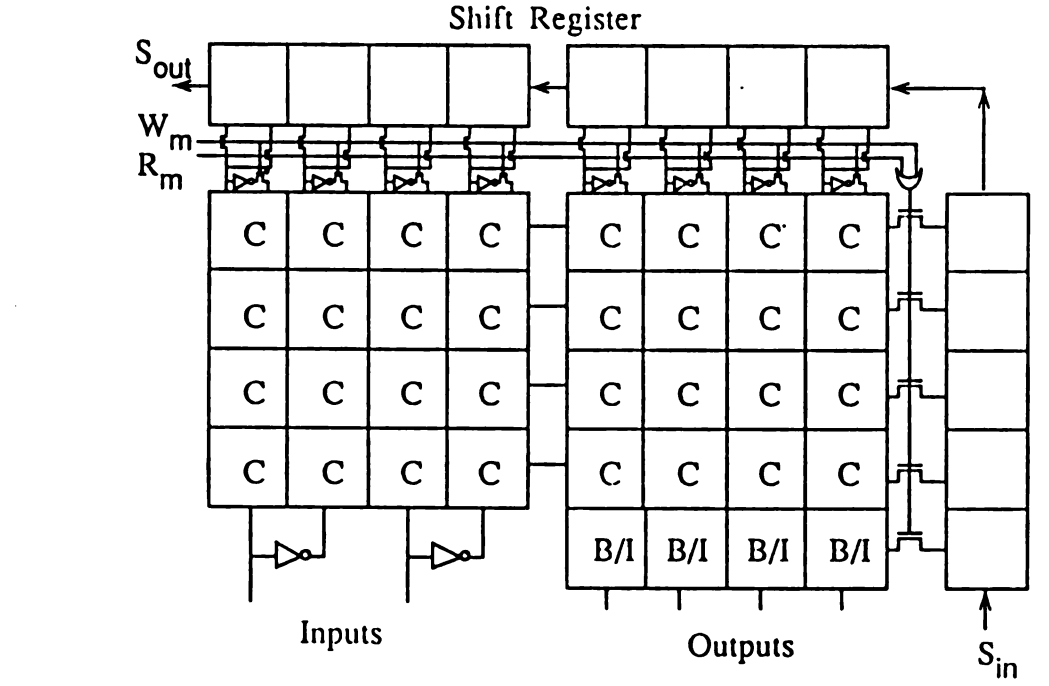
The control and refresh circuits and their operations can be found in [11,35] in detail. Here, the functions of the shift registers are discussed. Two sets of shift registers are connected to the AND and OR arrays. We first consider the two shift registers connected to the AND array. For simplicity, the shift register that closes to the R-cells is denoted as R-SR, while the other one is denoted as A-SR. During the programming phase [11,35], the R-SR contains data to program the A-cells, while the A-SR controls the column of A-cells to be programmed. More specifically, if the i-th cell of the A-SR holds the only logic 1, the A-cells in the i-th column are programmed with the data stored in the R-SR. Similarly, during the refreshing phase, while the A-SR enables the column to be refreshed, the R-SR is used to store the data. The shift registers connected to the OR array perform the same functions as discussed above. In summary, the shift registers are used for programming and refreshing the storage cells.

The shift registers also can be used in a similar fashion as the fault-tolerant PLA design discussed in Section 3.3 for fault detection and location. Based on the RBPLA of Figure 6.1 together with minor modification, a fault-tolerant DRBPLA design has been presented in [11], in which partially defective chips can be electrically repaired in field use.

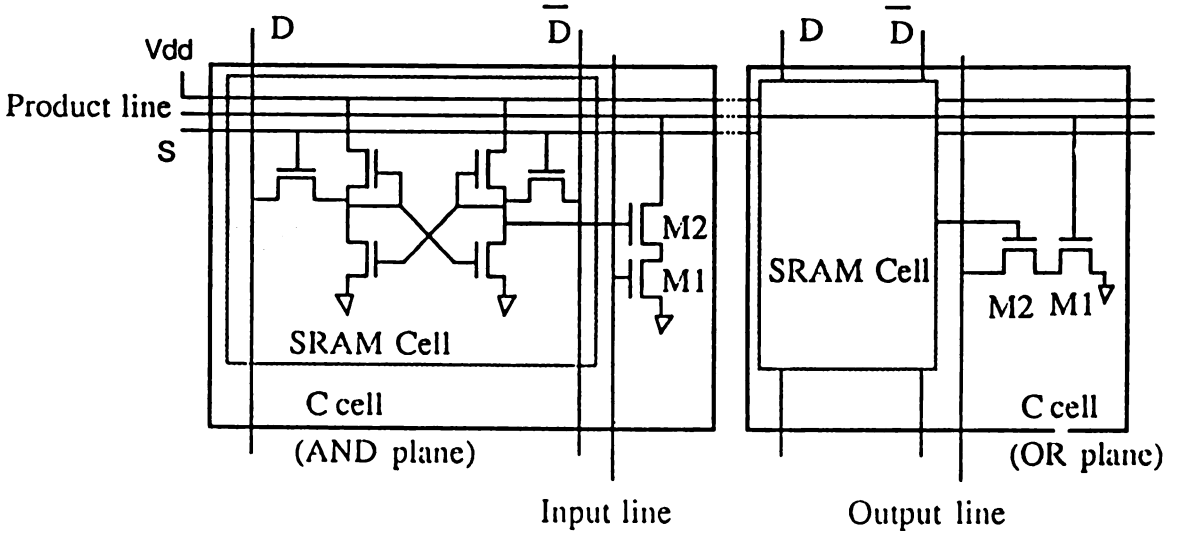
Because the structure and operation of the dynamic cell memory are much more complicated than those of the static cell memory, and also because both DRBPLA and SRBPLA have similarities in both fault models and fault-diagnosis process, for sake of clarity, the detailed SRBPLA design and its operation are preferably discussed in the next section. Fault models and fault-tolerant structure of SRBPLA design are presented in Section 6.2.

6.1.2 An SRBPLA Structure

Figure 6.2 illustrates the schematic diagram of a PLA structure based on the static cell memory. An SRBPLA consists of two major parts: (1) the basic PLA



(a)



(b) (c)

Figure 6.2 SRBPLA structure: (a) Scheme Diagram; (b) and (C) C cells in AND and OR plane.

functions with the AND and OR arrays, i.e., the C-cells (corresponding to crosspoints of a conventional PLA) and input/output lines; and (2) the control logic needed to program the logic of the PLA, such as shift registers, and read/write control circuits.

Figures 6.2 (b) and 6.2 (c) show the circuit schematic diagrams of the C-cells in the AND and OR arrays. The basic C-cell structure consists of an SRAM cell and transistors M1 and M2. The SRAM cell employs a pair of cross-coupled inverters as the storage flip-flop and a pair of access devices to provide a switchable path for data into and out of the cell. The cell enabled line S is held low except when cells connected to it are to be accessed for reading or for writing. Two lines, D and $\overline{D}$, provide the data path. The transistor M1 corresponds to the crosspoint in a conventional PLA. The presence (absence) of the crosspoint depends upon the transistor M2 in the ON (OFF) state which is determined by the state of the SRAM cell.

The SRBPLA allows designers to assign the output phase, i.e., the output may take either true or complementary logic realized by an non-inverting or inverting buffer, referred to as B/I-cell. The B/I-cell is normally in the inverting state (ON-state) unless it is programmed.

In addition to both arrays, three shift registers are also added with the control circuitry. For simplicity, the shift registers connected to AND-array and OR-array are referred to as AND-SR and OR-SR, respectively, while the vertical shift register is referred to as the S-SR. Figure 6.3 illustrates the control circuitry, where two signal lines W_m and R_m are used to control the "write" and "read" operations of the C cell, and the signal F is the output of the gate ORing W_m and R_m as shown. During the programming phase (or "write" operation), the shift registers AND-SR and OR-SR contain the data to program the C-cells in a specific row determined by the S-SR. More specifically, if the i-th cell of the S-SR holds the only logic 1, only the i-th row is enabled and thus the C-cells in the i-th row are programmed with the data held in

both AND-SR and OR-SR cells. The above operation is performed by setting W_m=1, and thus setting F=1 to pass the data held in the S-SR cell. On the other hand, during the "read" operation, the control signal R_m is set to logic 1 and results in F=1 to select the row of C-cells to be read. Finally, during the normal operation, both W_m and R_m are set to logic 0, and thus results in F=0, to isolate all three shift registers from the PLA.

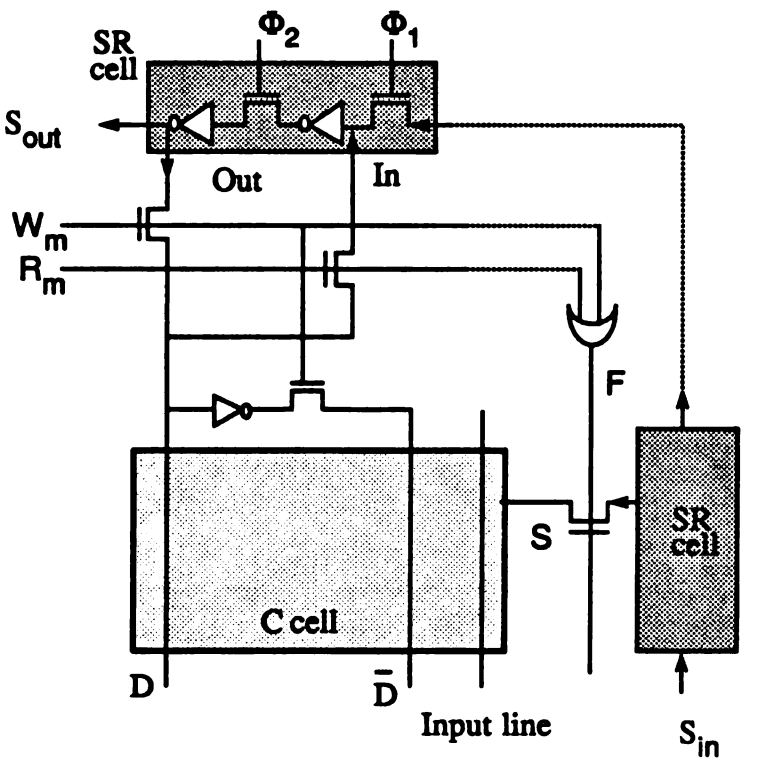


Figure 6.3 Control Circuit in SRBPLA.

6.2 A Fault-Tolerant SRBPLA Design

Similar to the fault-tolerant PLA design discussed in Section 3.3, the schematic diagram of a fault-tolerant SRBPLA is illustrated in Figure 6.4, where the original SRBPLA is augmented by SISC and SOSC, as well as the spare C-cells. In addition, a column of P-cells are also added in between the AND and the OR arrays. The P-cells are used to alleviate the problem for the repair of the *stuck-at*-1 faulty product line discussed in Section 3.2.

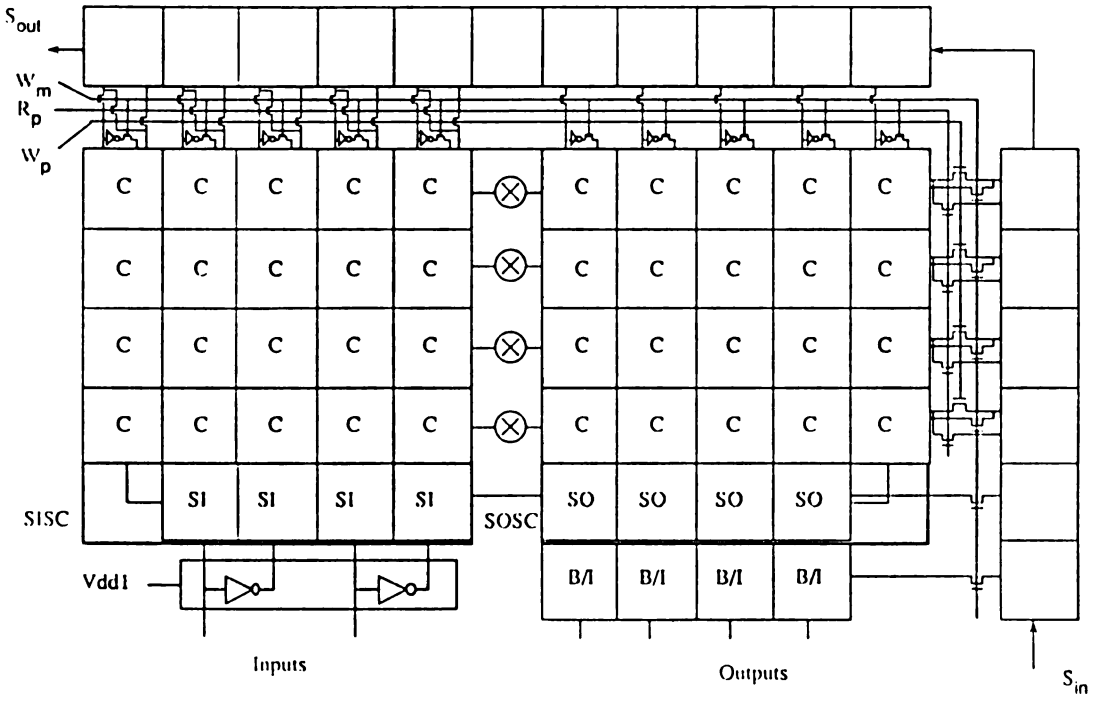


Figure 6.4 Fault-Tolerant SRBPLA Scheme.

The SISC and SOSC operations in the SRBPLA are operated in a similar fashion as those in RPLA discussed in Section 3.2. The basic cell structure of the SISC and its operation are illustrated in Figure 6.5. The basic cell is referred to as SI-cell. While Figure 6.5 (a) shows the NMOS implementation of an SI-cell, Figures

6.5 (b) and 6.5 (c) describe the operations of the SI-cell. During the normal operation, as shown in Figure 6.5 (b), (for simplicity, it is referred to as the ON state), a double throw switch is used to connect the input signal to the input line. The operation is equivalent to the case that the SRAM cell holds a 1 to turn on the transistor T1 and to turn off both transistors T2 and T3, as shown in Figure 6.5 (a). On the other hand, when a faulty input line is detected and located, the switch is reconfigured so that the input line is connected to the spare one and the faulty line is connected to Ground line. That is, the SRAM cell holds a logic 0 to turn off the transistor T1 and to turn on both T2 and T3. Similarly, the SO-cell, a basic cell of the SOSC, is operated exactly the same as the SI-cell

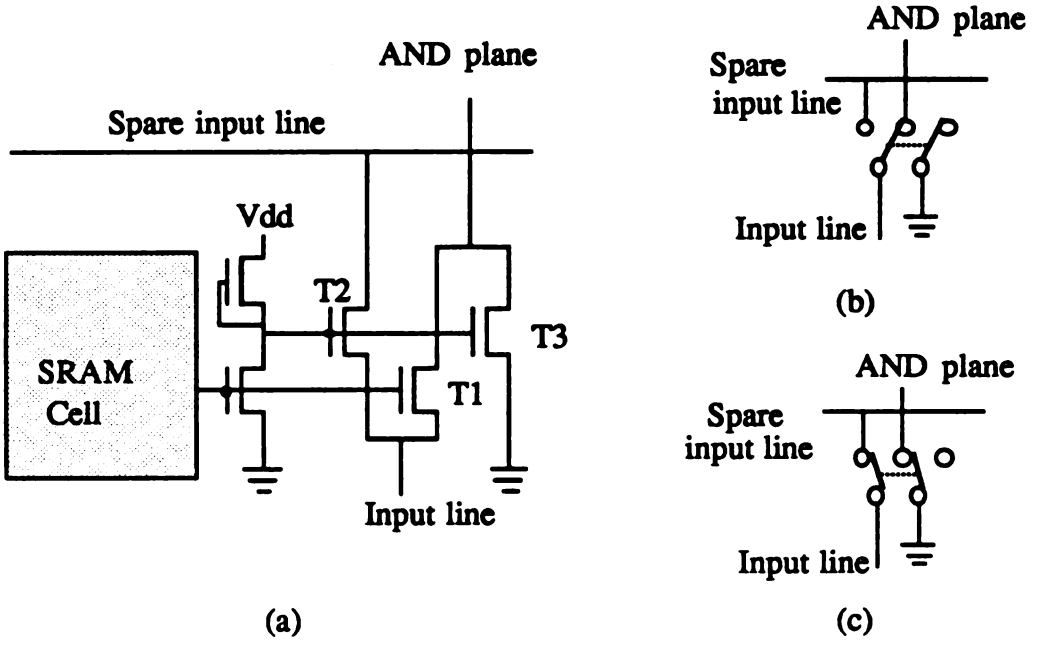


Figure 6.5 SI-Cell: (a) NMOS Implementation; (b) ON-state; and (c) OFF-state.

Figure 6.6 illustrates the circuit symbol and the basic cell structure of the product line link, referred to as P-cell, and its operation. The P-cell is programmed in the same way as the C-cell. When the SRAM cell holds a 1, Figure 6.6 (c) shows

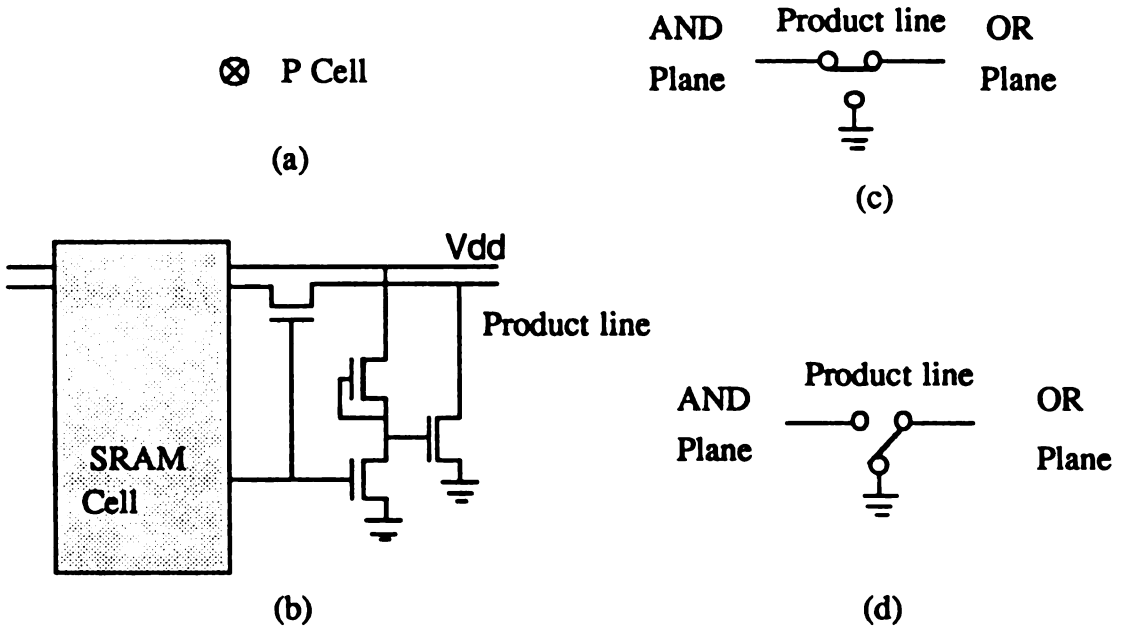


Figure 6.6 P-Cell: (a) Symbol; (b) NMOS Implementation; (c) ON-state; and (d) OFF-state.

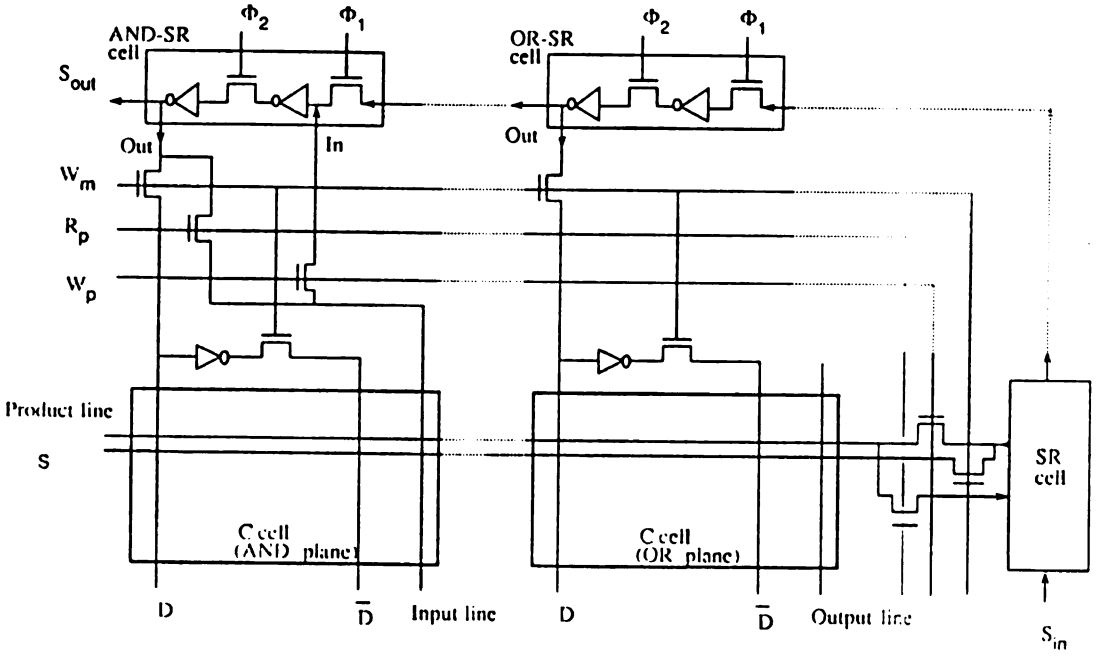


Figure 6.7 Control Signals of the Fault-Tolerant SRBPLA.

that the P-cell is turned to the ON (or Connecting) state during the normal operation. On the other hand, when the SRAM cell holds a 0, Figure 6.6 (d) describes that the P-cell is turned to the OFF (or Disconnecting) state if the faulty product line is located.

Figure 6.7 shows the schematic diagram of the control circuitry. Similar to the FDPLA of Figure 3.16, the internal signals R_p and W_p are used to control the operation of reading and writing the input and product lines.

6.3 Fault Diagnosis and Repair Process

This section describes the fault diagnosis and repair process of a fault-tolerant SRBPLA. The fault models considered here are also the *stuck-at*, *bridging*, and *crosspoint* faults. Based on the fault models, the SRBPLA is capable of detecting, locating, and repairing single and multiple faults during manufacturing process and in field use. Due to the similarity of the fault-tolerant designs for both SRBPLA and FDPLA, the fault models, repair rules, and diagnosis and repair process are almost the same. The differences are discussed below.

6.3.1 Fault Models

The major difference between the SRBPLA and FDPLA is in their crosspoint contacts. The SRBPLA takes the SRAM cell as its crosspoint contacts. In this implementation, any failures of the SRAM cell will affect the function of the crosspoint faults, and thus they are diagnosed as the crosspoint faults. Similarly, failures may occur at the SI-, SO-, and P- cells. However, if the failures do not affect the desired functions for those cells, then it is not necessary to locate them. For example, a fault-free product line requires the P-cell in the 1-state. Any failures that may cause product line to have a *stuck-at*-1, will not be concerned.

The failures diagnosed in the SI-cells (SO-cells) are considered as the input (output) lines having *stuck-at* faults. A *stuck-at*-1 faulty control signal S causes the corresponding row of C-cells to be read or written. On the other hand, a *stuck-at*-0 faulty signal results in the corresponding row of C-cells being at the idle state, i.e., no data are read and loaded.

Finally, since the adjacent lines in the SRBPLA are far apart to each other, the *adjacent bridging* faults will not be considered in SRBPLAs. That is, only *crossing bridging* faults are considered.

6.3.2 Fault Diagnosis and Repair Algorithm

Algorithm II, as illustrated in Appendix 3, summarizes the fault diagnosis and repair process for the fault-tolerant SRBPLA design. The process for SRBPLA is similar to that for FDPLA. The only differences are in the memory cells and other functional cells. After the shift registers are tested, as indicated in Step B.1, the state of each cell is initialized by setting 1's to the C-cells in the AND array and the spare output part, as well as the SI-, SO-, and P- cells, and 0's to the C-cells in the OR array. Then, the stuck-at faulty bit lines are located in the same way as discussed in FDPLA. In order to ensure that no s-a-1 faults occur in the S-lines, we try to write 0's to the C-cells in the AND array and 1's to the OR array. If the i-th S-line has a s-a-1 fault, the later diagnosis process will disclose that the C-cells in the i-th row contain all 0's instead of the expected 1's.

With the above programming in the C-cells, G- and A- faults, *stuck-at* faults at the product lines and output lines, and s-a-0 at the data line of C-cells can be diagnosed. However, in order to identify the S- and D- faults, s-a-1 faults at the data line, and s-a-0 faults in the product lines and output lines, the contents of the C-cells are programmed by their complemented states as shown in Step E.1.

6.4 Summary

A RAM-based PLA structure is presented to allow designers to reprogram the logic functions as many times as needed. A fault-tolerant SRBPLA design is also proposed that can detect, locate, and repair faults in the manufacturing process, and also in field use. A fault-diagnosis and repair algorithm gives the evidence that the proposed fault tolerant design achieves a full diagnosability of single and multiple *crosspoint* faults, *bridging*, faults, and *stuck-at* faults.

CHAPTER 7

Conclusions

This chapter summarizes the major contribution of this dissertation research and outlines directions for future research.

7.1 Summary of Major Contributions

The yield of integration circuits (ICs) has always been crucial to the commercial success of their manufacture. The manufacturing of VLSI products with increasing complexity is the current trend of IC market. However, as the complexity increases and the geometry shrinks, the probability of having faulty components also increases, thereby lowing the chip yield.

One practical way to solve the low yield problem is the use of fault-tolerant design. The only integrated circuits so far to have exploited fault-tolerant techniques commercially have been memory chips. This is because memory chips are particularly densely packed and therefore increasingly vulnerable to defects, and also because a regular memory array lends itself to a variety of efficient fault-tolerant designs.

Recently, due to the structural regularity, design simplicity, and fast turnaround time, Programmable Logic Arrays (PLAs) have been increasingly popular for implementing Boolean logic functions for VLSI/ULSI chips. Due to the similar structure, large PLAs are also increasingly vulnerable to defects. In order to repair partially defective PLA chips to enhance overall chip yield, a fault-tolerant PLA design is proposed in this dissertation. Faults in a partially defective PLA chip can

be detected, located, and repaired. The repair does not affect the external signal routing. It is the first fault-tolerant structure for PLA. Included in the study are the structures of repairable PLAs, and fault-diagnosable PLAs, an automatic layout generator for fault-tolerant PLAs, and a fault diagnosis and repair process.

Some important issues for the design of fault-tolerant design such as die size, speed, and yield enhancement, have also been addressed in this study. The results of this study have shown that the yield can be enhanced significantly for large PLAs. Another important issues in optimal redundancy and spare allocation have also been discussed in Chapter 5. A simple, yet efficient optimization method has been presented to determine the optimal redundancy of various sizes of PLAs.

Finally, a RAM-based PLA structure is proposed with its fault-tolerant design. Although the PLA structure taking RAM cells as its crosspoint contacts consumes more silicon area than the conventional PLA, its re-programmability allows the designers to change the design as many times as needed. Exploiting the advantages of the shift registers used for programming memory cells in the RAM-based PLA structure, the shift registers are also used to detect and locate faults that occur either in manufacturing processing, or in field use. In Chapter 6, the fault-tolerant SRBPLA design is presented with the fault-diagnosis and repair process.

In summary, this dissertation focuses on fault-tolerant design of large PLAs. It is novel in the sense that it is the first fault-tolerant structure ever proposed for PLAs. This structure will be helpful in achieving fault diagnosability and repair, and thus improving chip yields.

7.2 Directions for Future Research

Array logic has been used extensively for structured design of LSI systems. In particular, the PLA has been proved to be a very effective tool for implementing multiple output combinational logic functions. In this section, some practical issues in

the fault-tolerant design for yield enhancement in array logic are addressed for further study.

7.2.1 Fault-Tolerant Design of Folded PLAs

The PLA is an architectural structure used to reduce the design effort in producing VLSI integrated circuits. However, industrial PLAs tend to be sparse and thus grossly wasteful of chip area [34], i.e., area occupied only by interconnections and not directly contributing to the implementation of the logic functions. The wasted area will reduce circuit yield while also degrading the time performance of the PLA by introducing unnecessary parasitics [21]. Folding is a technique which attempts to reduce the area of a PLA by exploiting its sparsity. In other words, the objective of a folding technique is to determine permutations of the rows (and/or columns) which permit a maximal set of column pairs (row pairs) to be implemented in the same column (row) of the physical logic array. There are many kinds of folding according to the particular technological implementation and design style followed, e.g., Simple and Multiple Column Folding, Simple and Multiple Row Folding, and Constrained Folding. The results of the study in [21] have shown that the chip area can be reduced as much as 50% if a simple folding technique is applied.

Although the folded PLAs significantly reduce the chip area from the original PLA, the test of such PLAs is very difficult because the availability of the testable structure, thereby limiting its applicability. Also, a "cut" is generally applied to a row (column) shared by a row pair (column pair). The cut may introduce a *bridging* fault in that row (column), i.e., the line is not completely "cut". Thus, the low yield problem has been found in the folded PLA design. Therefore, the marriage of fault-tolerant design and folding technique is perfectly applied to enhance the chip yield. This leads to a very interesting topic for future research.

7.2.2 Fault-Tolerant Design of VLSI/ULSI/WSI Array Structures

Wafer scale integration (WSI) offers advantages of low assembly cost, high reliability, and high performance. However, a major problem of the WSI approach is its low manufacturing yield. Without redundant components and a reconfiguration mechanism for replacing defective components, the yield of a wafer scale device is very likely to approach zero. Therefore, it is necessary to build redundancy and reconfiguration mechanisms into a WSI device for improving the yield [61].

Two-level redundancy scheme has been discussed in [61]. While the first level of redundancy, called the local redundancy, can repair the point defects, the second level of redundancy, called the global redundancy, repairs the cluster defects. Memory devices with two-level redundancy scheme have been widely used in the WSI systems. The point defects occurred at the cell array can be repaired by the spare cells (the use of local redundancy). On the other hand, those point defects occurred at non-redundant areas, as well as the cluster defects, will cause the redundant memory to be unrepairable, and this unrepairbale device is thus replaced by the other fault-free device (the use of global redundancy).

It has been shown that a two-level redundancy scheme can improve the manufacturing yield significantly. The implementation of PLA as controller in WSI has not yet been considered so far. This is simply because the repair of PLA was not possible. With the success of this study, the proposed fault-tolerant PLA structure can be applied to the design of the WSI system to improve the manufacturing yield. This leads to another research topic for further study.

Due to the regular structures, the fault-tolerant design of VLSI array structures such as memory and PLA, have been considered. The next logical step is to implement the proposed fault-tolerant design for the design of some other array structures, such as Storage Logic Arrays (SLAs) [51], or Programmable Array Logics (PALs) [14].

As the technology moves to ULSI, more than one million transistors can be incorporated on a single chip. Regularity plays an important role in the ULSI

approach. The fault-tolerant design is a practical way to enhance the chip yield before the manufacturing process is matured.

7.2.3 New Yet Low-Yield Technologies

As mentioned previously, it is desirable to incorporate redundant and fault-tolerance process into any regular structures with low yield problems. In general, the manufacturing yield for a new technology is generally low. It has been reported that yield of LSI GaAs circuits is very low when the technology is applied to memory design [26]. Table 7.1 lists the overall yield drops from 52% to 1% for the JFET technology as the RAM size increases from 1K to 16K. Recently, several manufacturers have tried to produce programmable logic devices implementing GaAs technology. Unfortunately, the yields are unreasonably low. Therefore, fault-tolerant structures should be applied to those regular structures with new technology.

Table 7.1 Yields of LSI GaAs Circuits [26]

Circuit	Best Wafer	Overall lot	Size
D-MESFET			
16 K RAM	2%	0.5%	7.3 × 7.5 mm
4 K RAM	40%	15%	3.4 × 7.5 mm
1 K RAM	55%	8%	N/A
JFET			
16 K RAM	3%	1%	8.2 × 5.6 mm
4 K RAM	41%	16%	4.1 × 5.0 mm
1 K RAM	71%	52%	2.6 × 2.5 mm

APPENDICES

APPENDIX 1

Propagation Delay Time

As mentioned in Chapter 3, the propagation delay is based on the assumption that the delay time of a logic gate is directly related to the driving capability of its transistor. Under the above assumption, the delay time for a non-redundant PLA with n inputs, p product terms, and m outputs, namely, a (n,p,m)-PLA, is estimated. The total effective load capacitance in the AND plane is comprised of the total gate capacitance in an input line of the AND plane and the worst case signal path capacitance. The total gate capacitance depends on the number of pull-down transistor in an input line. Suppose that the number of pull-down transistors in an input line is g, the total gate capacitance is gC_g. The worst case signal path capacitance is pC_p because there are p product lines across to the input lines. Therefore, the total delay time in the AND plane is approximately estimated as

$$T_n = \kappa \{ (2n+p) C_p + g C_g \}$$

where κ is a constant determined by the average charging time.

For the redundant PLA of Figure 3.15 (a), the worst case signal path includes the path to utilize the spare input lines. Therefore, the worst case signal path capacitance is $(4n+p)C_p$, and the delay time is about

$$T_r = \kappa \{ (4n+p)C_p + g C_g \}$$

In fact, the number of pull-down transistor in the PLA design is determined by the function implemented. The distribution of the locations of the pull-down transistor is usually spare. Therefore, it is reasonable to assume that the average number of pull-down transistors in a column of the AND plane is about the half of the number of product lines, i.e. $g=p/2$. The quantities of the gate and signal path capacitances have

been studied and approximately estimated as $C_g = 1.77 \times 10^{-14}$ F and $C_p = 0.6084 \times 10^{-15}$ F, i.e., $C_g = 29C_p$. Thus, the ratio of the delay times for the PLA with and without redundancy in the AND plane is

$$T_r / T_n = 1 + 2n / (2n+15.5p)$$

APPENDIX 2

Fault Diagnosis Algorithm for FDPLA

Algorithm I:

* n: number of input lines; s_n: number of spare bit lines;
* m: number of output lines; s_m: number of spare output lines;
* p: number of product lines; s_p: number of spare product lines.
* r: number of input bit lines; r=2n.
*

Step A. /* Test the added circuits */

Set R=W=0 to isolate both ISR and PSR from the PLA;
Apply a scan pattern to detect stuck-at faults;
If any faults are detected, GOTO Step UR. /* Un-repairable */

Step B. /* Stuck-at and bridging faults at the input bit lines */

B.1. Set Vdd1=1, (R,W)=(0,1), /* ISR - "read" mode */
M_P=0, and shift all 0's to PSR. /* apply all 0's to P_i's */

B.2. Apply patterns (I_1,..,I_n)=(0,..,0), and (1,..,1) to obtain EP0 and EP1.

B.3. Flag:=0;
For i=1 to n Do
If (EP0(i)=EP1(i)) Then /* Stuck-at faults at b_i */
Begin
if (s_n=0) Then GOTO Step UR; /* Un-repairable */
Repair b_i by a spare input bit line;
Flag:=1;
s_n:=s_n-1;
End
If (Flag=1) GOTO Step B.2. /* Check faults in spare lines */

Step C. /* G-faults, S-faults, and Stuck-at faults at the product lines */

C.1. Set Vdd1=0, (R,W)=(1,0). /* ISR - "write" mode; PSR - "read" mode */
Set M_I=0 and shift all 0's to ISR. /* apply all 0's to b_i's */

C.2. Flag:=0;
For i=1 to p Do
If (P_i=0) Then /* s-a-0 fault at P_i */
Begin
If (s_p=0) GOTO Step UR; /* Un-repairable */
Repair P_i by a spare product line;
Flag:=1;
s_p:=s_p-1;
End
If (Flag=1) GOTO Step B. /* Check faults in spare lines */

C.3. /* Generate the matrix AP_F */
For j=1 to r Do
Begin
Set b_j=1 and the others to 0's;
For i=1 to p Do A(i,j):= complement of P_i;
End

C.4. Flag:=0;
For i=1 to p Do
If (the i-th row of AP_F contains all 0's) Then
Begin /* s-a-1 fault at P_i */
If (s_p=0) GOTO Step UR; /* Un-repairable */
Repair P_i by a spare product line;
Flag:=1;
s_p:=s_p-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */

C.5. /* Generate the matrix CAP */
For i=1 to p
For j=1 to r
CAP(i,j):=a(i,j) $\oplus$ A(i,j);

Step D. /* D-faults, A-faults, Stuck-at and bridging faults at the output lines, and bridging faults at the product lines*/

D.1. Set Vdd1=0, (R,W)=(0,1). /* ISR - "read" mode; PSR - "write" mode */
Set M_P=0 and shift all 0's to PSR. /* apply all 0's to P_i's */

D.2.
```
Flag:=0;
For j=1 to m Do
If (Oj=1) Then  /* s-a-1 fault at Oj */
Begin
    If (sm=0) GOTO Step UR;  /* Un-repairable */
    Repair Oj by a spare output line;
    Flag:=1;
    sm:=sm-1;
End
If (Flag=1) GOTO Step B. /* Check faults in spare lines */
```

D.3.
```
/* Generate the matrix OPF */
For j=1 to p Do
Begin
    Set Pj=1 and the others to 0's;
    For i=1 to m Do B(j,i):=Oj;
End
```

D.4.
```
Flag:=0;
For i=1 to m Do
If (the i-th column of OPF contains all 0's) Then
Begin  /* s-a-0 fault at Oi */
    If (sm=0) GOTO Step UR; /* Un-repairable */
    Repair Oi by a spare output line;
    Flag:=1;
    sm:=sm-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */
```

D.5. Flag:=0;
For i=1 to p Do
If (the i-th row of OP_F contains all 0's) Then
Begin /* s-a-0 fault at P_i */
If (s_p=0) GOTO Step UR; /* Un-repairable */
Repair P_i by a spare product line;
Flag:=1;
s_p:=s_p-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */

D.6. /* BROO bridging faults */
Flag:=0;
If (bit patterns of w adjacent columns in OP_F are the same) Then
If (the bit pattern equals to the pattern of ANDing the corresponding columns in OP) Then /* BROO Bridging faults */
Begin
Flag:=1;
s_m:=s_m-w;
If (s_m < 0) GOTO UR; /* Un-repairable */
Repair these w's bridged lines by spare output lines.
End
Repeat this step until all columns are checked for BROO faults.
If (Flag=1) GOTO Step B. /* Check faults in spare lines */

D.7. /* Generate the matrix COP */
For i=1 to p
For j=1 to m
COP(i,j):=b(i,j) $\oplus$ B(i,j);

Step E. /* Spare allocation for crosspoint faults */
Call spare allocation routine to repair crosspoint faults.
If (the required spares do not exceed the available spares) Then
the defective chip is repairable and STOP.

Step UR. /* Un-repairable */
The defective chip is not repairable.

APPENDIX 3

Fault Diagnosis Algorithm for SRBPLA

Algorithm II:

Step A. /* Test the shift registers */

Same as part A of Algorithm I, except setting $(R_p,W_p,W_m)=(0,0,0)$.

Step B. /* Identify Stuck-at and bridging faults at the input bit lines */

B.1. /* Initialize the state of the SRBPLA */

Set $(R_p,W_p,W_m)=(0,0,1)$, and S-SR=(1,1,...,1);
Load 1's to the C-cells in the AND arrays, i.e.,
the personality of matrix AP contains all 1's entries;
Load 0's to the C-cells in the OR arrays, i.e., OP(i,j)=0;
Load 1's to the C-cells of the spare output lines; and
Load 1's to the SI-, SO-, P-cells.

B.2. - B.4. /* Locating the stuck-at faults at b_i */

Same as B.1. - B.3 of Algorithm I except setting $(R_p,W_p,W_m)=(0,1,0)$

Step C. /* Identify G-faults, and stuck-at faults at the product */

C.1. Set $(R_p,W_p,W_m)=(0,0,1)$, S-SR=(0,0,...,0); AND-SR=(0,0,...,0), and OR-SR=(0,0,...,0); /* A stuck-at-1 fault at j-th S signal will re-program the j-th row C-cells from 1 to 0 in the AND plane */

C.2. - C.5. /* Identify stuck-at faults at the product lines and generate matrix CAP */

Same as C.1. - C.4. of Algorithm I except setting $(R_p,W_p,W_m)=(1,0,0)$

C.6. /* Identify s-a-0 fault at the data path */
Flag:=0;
For i=1 to r Do
If (the i-th column of AP_F contains all 0's) Then
Begin /* i-th column has s-a-0 fault at signal D */
If (s_n=0) GOTO Step UR; /* Un-repairable */
Repair b_i by a spare input line;
Flag:=1;
s_n:=s_n-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */

C.7. /* Generate the matrix CAP */

Same as C.5. of Algorithm I

Step D. /* Identify A-faults, s-a-1 and bridging faults at the output lines, and bridging faults at the product lines)

D.1. - D.3. /* Generate matrix OP_F and identify s-a-1 faults at O_i */

Same as D.1. - D.3. of Algorithm I except setting (R_p,W_p,W_m)=(0,1,0)

D.4. /* Stuck-at fault at O_i */
Flag:=0;
For i=1 to m Do
If (the i-th column of OP_F contains all 1's) Then
Begin
If (s_m=0) GOTO Step UR; /* Un-repairable */
Repair O_i by a spare output line;
Flag:=1;
s_m:=s_m-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */

D.5. /* Generate the matrix COP */

Same as D.5. of Algorithm I

Step E. /* S- and D-faults, s-a-0 at outputs */

E.1. /* Re-program the state of the AND and OR arrays */
Set (R_p,W_p,W_m)=(0 ,0,1), and S-SR=(1,1,...,1);
Load 0's to the C-cells in the AND arrays, i.e., AP(i,j)=0;
Load 1's to the C-cells in the OR arrays, i.e., OP(i,j)=1;

E.2. /* Generate the matrix AP_F */

Same as C.3. of Algorithm I

E.3. /* Identify s-a-1 faults at the data path */
Flag:=0;
For i=1 to r Do
If (the i-th column of AP_F contains all 1's) Then
Begin /* i-th column has s-a-1 fault at signal D */
If (s_n=0) GOTO Step UR; /* Un-repairable */
Repair b_i by a spare input line;
Flag:=1;
s_n:=s_n-1;
End
If (Flag=1) GOTO Step B; /* Check faults in spare lines */

E.4. /* Generate the matrix CAP */
For i=1 to p
For j=1 to r
CAP(i,j):= CAP(i,j) + (a(i,j) ⊕A(i,j));

E.5. - E.7. /* Generate the matrix OP_F and
identify s-a-0 faults at the product and outputs */

Same as D.3. - D.5. of Algorithm I

E.8. /* Generate the matrix COP_F */
For i=1 to p
For j=1 to m
COP(i,j):= COP(i,j) + (b(i,j) ⊕B(i,j));

Step F. /* Spare allocation for crosspoint faults */
Call spare allocation routine to repair crosspoint faults.
If (the required spares do not exceed the available spares) Then
the defective chip is repairable and STOP.

Step UR. /* Un-repairable */
The defective chip is not repairable.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Abbott, R, K. Kokkonen, R. I. Kung, and R. J. Smith, "Equipping a Line of Memories with Spare Cells," Electronics, pp. 127-130, July 28, 1981.

[2] Abraham, J. A., and W. K. Fuchs "Fault and Error Models for VLSI," IEEE Proceedings, Vol. 74, No. 5, pp. 639-654, May 1986.

[3] Albuquerque, N. M., "Cut-and-Patch Lasers Speed Chip Repairs," Electronics, pp. 19-20, June 16, 1986.

[4] Asai, S., "Semiconductor Memory Trends," IEEE Proceedings, Vol. 74, No. 12, pp. 1623-1635, Dec. 1986.

[5] Bindels, J. F. M., J. D. Chlipala, F. H. Fisher, T. F. Mantz, R. G. Nelson, and R. T. Smith, "Cost-Effective Yield Improvement in Fault-Tolerant VLSI Memory," IEEE International Solid State Circuits Conference Digest, New York, NY, pp. 82-83, Feb. 1981.

[6] Bozorgui-Nesbat, S., and E. J. McCluskey, "Lower overhead design for testability of programmable logic arrays," IEEE Trans. on Computers Vol. C-35, No. 4, pp. 379-383, April 1986.

[7] Cenker, R. P., D. G. Clemons, W. R. Huber, J. B. Petrizzi, F. J. Procyk, and Trout G. M, "A Fault-Tolerant 64k Dynamic RAM," IEEE Trans. on Electron Devices, Vol. ED-26, No. 6, pp. 853-860, June 1979.

[8] Chang, M.-F., W. K. Fuchs, and J. H. Patel, "Diagnosis and Repair of Memory with Coupling Faults," Proceeding of the International Conference on Computer-Aided Design, Santa Clara, CA, pp. 524-527, Nov. 1988.

[9] Chang, T. Y., "The Design of Testable Programmable Logic Arrays," Master thesis, Michigan State University, 1987.

[10] Chang, T. Y., "*MRPLA* Users Manual," Dept. of Electrical Engineering, Michigan State University, 1989.

[11] Chang, T. Y., and C. L. Wey, "The Design of Electrically Field-Repairable Programmable Logic Arrays (EFRPLAs)," Proceeding of 31st Midwest Symposium on Circuits and Systems, pp. 36-39, St. Louis, MO., Aug. 1988.

[12] Chang, T. Y., and C. L. Wey, "Design of Fault Diagnosable and Repairable PLA," IEEE Journal of Solid State Circuits, Vol. SC-24, No. 5, pp. 1451-1454, Oct. 1989.

[13] Day, J. R., "A Fault-Driven Comprehensive Redundancy Algorithm for Repair of Dynamic RAMs," IEEE Design and Test of Computers, Vol. 2, No. 3, pp.35-44, June 1985.

[14] *Design with Programmable Array Logic*, Technical Staff of Monolithic Memories, Inc., New York, McGraw-Hill, 1981.

[15] *EP300, Erasable Programmable Logic Devices*, Altera Corp., Santa Clara.

[16] Fawcett, B. K., "Taking Advantage of Reconfigurable Logic," Programmable Logic Guide, High Performance, pp. 17-24, 1989.

[17] Ferry, D., L. A. Akers, and E. W. Greeneich, *Ultra Large Scale Integrated Microelectronics*, Prentice Hall Advanced Reference Series, 1988.

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

[19] Fong, E., M. Converse and P. Denham, "An Electrically Reconfigurable Programmable Logic Array Using a CMOS/DMOS Technology," IEEE Journal Solid State Circuits, Vol. SC-19, No. 12, pp. 1041-1043, Dec. 1984.

[20] Fucks, W. K. ,and S. Y. Kuo, "Spare Allocation/Reconfiguration for WSI," *Wafer Scale Integration*, Ed. by E.E. Swartzlander, Jr., Kluwer Academic Publishers, pp. 119-191, 1989.

[21] Hachtel, G. D., A. R. Newton, and A. L. Sangiovanni-Vincentelli, "An Algorithm for Optimal Folding," IEEE Trans. on CAD, Vol. CAD-1, No. 2, pp. 63-76, April 1982.

[22] Haddad, R. W., and A. T. Dahbura, "Increased Throughput for the Testing and Repair of RAMs with Redundancy," Proceeding of the International Conference on Computer-Aided Design, Santa Clara, CA, pp. 230-233, Nov. 1987.

[23] Hasan, N., and C. L. Liu, "Minimum Fault Coverage in Reconfigurable Arrays," Proceeding IEEE International Fault-Tolerant Computing Symposium, pp. 348-353, 1988.

[24] Hemmady, V. G., and S. M. Reddy, "On the Repair of Redundant RAMs," Proceeding 26th ACM/IEEE Design Automation Conference, Anaheim, CA, pp. 710-713, June 1989.

[25] Khakbaz, J., "A Testable PLA Design with Low Overhead and High Fault Coverage," IEEE Trans. on Computers, Vol. C-33, No. 8, pp. 743-745, Aug. 1984.

[26] Kirkpatrick, C. G., "Making GaAs Integrated Circuits," IEEE Proceedings, Vol. 76, No.7, pp. 792-815, July 1989.

[27] Hnatek, E. R., *Semiconductor Memories - an update*, PennWell Publishing Company, 1982.

[28] Koren, I., "A Reconfigurable and Fault-tolerant VLSI Multiprocessor Array," Proceeding 8th Symposium Computer Architecture, pp. 442-451, May 1981.

[29] Koren, I., and D. K. Pradhan, "Modeling the Effect of Redundancy on Yield and Performance of VLSI Systems," IEEE Trans. on Computers, Vol. C-36, No. 3, pp. 344-355, March 1987.

[30] Kuo, S. Y. and W. K. Fuchs, "Efficient Spare Allocation in Reconfigurable Arrays," IEEE Design and Test of Computers, pp. 24-31, Feb. 1987.

[31] Kuo, S. Y., and W. K. Fuchs, "Fault Diagnosis and Spare Allocation for Yield Enhancement in A Large Reconfigurable PLAs," Proceeding IEEE International Test Conference, Washington DC, pp. 944-951, Sep. 1987.

[32] Law, H.-F. S., and M. Shoji, "PLA Design for BELLMAC-32A Microprocessor," Proceeding International Conference Circuits and Computers, pp. 161-164, Feb. 1982.

[33] Mak, G. P., J. A. Abraham, and E. S. Davidson, "The Design of PLAs with Concurrent Error Detection," Proceeding IEEE International Test Conference, Philadelphia, PA, pp. 303-310, Nov. 1982.

[34] Makarenko, D. D., J. Tartar, "A Statistical Analysis of PLA Folding," IEEE Trans. on CAD, Vol. CAD-5, No. 1, pp. 39-51, Jan. 1986.

[35] Marchand, J. F. P., "An Alterable Programmable Logic Array," IEEE Journal Solid State Circuits, Vol. SC-20, No. 5, pp. 1061-1066, Oct. 1985.

[36] Mayo, R. N., and J. K. Ousterhout, "Pictures with Parentheses: Combining Graphics and Procedures in a VLSI layout Tool," Proceeding 20th ACM/IEEE Design Automation Conference, Miami Beach, FL, pp. 270-276, June 1983.

[37] Mead, C. A., and L. A. Conway, *Introduction to VLSI systems*, Addison-Wesley, Reading, Mass. 1980.

[38] Minato, O., T. Masuhara, T. Sasaki, Y. Sakai, T. Hayashida, K. Nagasawa, K. Nishimurs, "A Hi-CMOS 8k x 8k bit static RAM," IEEE Journal of Solid State Circuits, Vol. SC-17, No. 5, pp. 793-797, Oct. 1982.

[39] Moore, W. R., "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," IEEE Proceedings, Vol. 74, No. 5, pp. 684-698, May 1986.

[40] *MPLA User's Manual,* Berkeley VLSI Tools. Department of Electrical Engineering and Computer Science, University of California at Berkeley.

[41] Ostapko, D. L., and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-28, No. 9, pp. 617-626, Sept. 1979.

[42] Posa, J. G.,"What to Do When the Bits Go Out," Electronics, pp. 117-120, July 28, 1981.

[43] Pradhan, D. K., and K. Son, "The Effects of Untestable Faults in PLAs and a Design for Testability," Proceeding IEEE International Test Conference, Philadelphia, PA, pp. 359-367, Nov. 1980.

[44] Sack, E. A., R. C. Lyman, and G. Y. Chang, "Evolution of the concept of a computer on a slice," IEEE Proceedings, Vol. 52, pp. 1713-1779, Dec. 1964.

[45] Sasaki, H., "Directions and Strategies to Achieve ULSI," VLSI Technology Digest, San Diego, CA, pp. 3-6, May 1988.

[46] Schuster, S. E., "Multiple Word/Bit Line Redundancy for Semiconductor Memories," IEEE Journal Solid-State Circuits, SC-13, No.5, pp. 698-703, Oct. 1978.

[47] Sear, D., "Is Bigger Always Better?," ASIC Technology & News, pp. 14-16, May 1989.

[48] Shen, J., W. Maly, and F. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuit," IEEE Design and Test of Computers, pp. 13-26, Dec. 1985.

[49] *Signetics Field Programmable Logic Arrays*, Signetics, CA, Oct. 1977.

[50] Smith, J. E., "Detection of Faults in Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-28, No. 11, pp. 845-853, Nov. 1979.

[51] Smith, K. F., T. M. Carter, and C. Hunt, "Structured Logic Design of Integrated Circuits Using the Storage/Logic Array (SLA)," IEEE Trans. on Electron Device, Vol. ED-29, No. 4, pp. 765-776, April 1982.

[52] Smith, R. S., J. D. Chlipala, J. F. M. Bindels, R. G. Nelson, F. H. Fisher, and T. F. Mantz, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," IEEE Journal of Solid State Circuits, Vol. SC-16, No. 5, pp. 506-513, Oct. 1981.

[53] Smith, R. T., "Using a Laser Beam to Substitute Good Cells for Bad," Electronics, pp. 131-134, July 28, 1981.

[54] Somenzi, F., and S. Gai, "Fault Detection in Programmable Logic Arrays," IEEE Proceedings, Vol. 74, No. 5, pp. 655-668, May 1986.

[55] Stapper, C. H., A. N. McLaren and M. Dreckmann, "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product," IBM Journal Research Development, Vol. 24, pp. 398-409, May 1980.

[56] Stewart, D. M., "Production Test and Repair of 256k Dynamic RAMs with Redundancy," Proceeding IEEE International Test Conference, Philadelphia, PA, pp. 471-474, Oct. 1983.

[57] Strojwas, A. J., "Design for Manufacturability and Yield," Proceeding 26th ACM/IEEE Design Automation Conference, Las Vegas, NV, pp. 454-459, June 1989.

[58] Tarr, M., D. Boudreau, and R. Murphy, "Defect Analysis System Speeds Test and Repair of Redundant Memories," Electronics, pp. 175-179, Jan. 12, 1984.

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

[60] Walker, D. M. H., *Yield Simulation for Integrated Circuits*, Kluwer Academic Publishers, Boston, 1987.

[61] Wang, M., M. Culter, and S. Y. H. Su, "Reconfiguration of VLSI/WSI Mesh Array Processors with Two-Level Redundancy," IEEE Trans. on Computers, Vol. C-38, No. 4, pp. 547-554, April 1989.

[62] Wey, C. L., "On yield Considerations for the Design of Redundant Programmable Logic Arrays," IEEE Trans. on CAD, Vol. CAD-7, No. 4, pp. 528-535, April 1988.

[63] Wey, C. L., and F. Lombardi, "On the Repair of Redundant RAM's," IEEE Trans. on CAD, Vol. CAD-6, No. 2, pp. 222-231, March 1987.

[64] Wey, C. L., T. Y. Chang and M. K. Vai, "On the Design of Fault-Tolerant Programmable Logic Arrays," Proceeding International Computer Symposium, Taiwan, pp. 298-304, Dec. 1986.

[65] Wey, C. L., M. K. Vai, and F. Lombardi, "On the Design of a Redundant Programmable Logic Array (RPLA)," IEEE Journal of Solid-State Circuits, Vol. SC-22, No.1, pp. 114-117, Feb. 1987.